Міністерство освіти і науки України
Криворізький національний університет
Факультет інформаційних технологій
Кафедра комп’ютерних систем та мереж

Пояснювальна записка
до кваліфікаційної роботи бакалавра
за спеціальністю 123 «Комп’ютерна інженерія»

на тему: СИСТЕМА КАТАЛОГІЗАЦІЇ СТУДЕНТСЬКИХ РОБІТ

Проектував _____

Керівник роботи _____

Консультант _____

Нормоконтроль _____

Завідувач кафедри _____

Кривий Ріг
2025

Криворізький національний університет
Факультет інформаційних технологій
Кафедра комп'ютерних систем та мереж

Ступінь вищої освіти бакалавр
Спеціальність 123 «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ
Завідувач кафедри, голова циклової комісії

_____ А. І. Купін

"____" _____ 20__ року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

_____
(прізвище, ім'я, по батькові)

1. Тема роботи _Система каталогізації студентських робіт_____
_____
_____
керівник роботи _____,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від "___"_____ 20__року №__
2. Строк подання студентом роботи _____
3. Вихідні дані до роботи _____
_____
_____
_____
_____
_____
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)_____
_____
_____
_____
_____
_____
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)_____
_____
_____
_____
_____

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання_____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Строк виконання етапів роботи	Примітка

Студент _____ _____
(підпис) (прізвище та ініціали)

Керівник роботи _____ _____
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка: 55 сторінок, 22 рисунки, 6 таблиць, 1 додаток, 35 використаних джерел.

Об’єкт аналізу – система каталогізації студентських робіт, призначена для централізованого управління академічними проєктами у вищих навчальних закладах.

Проєкт складається з двох основних розділів.

Перший розділ присвячений аналізу вимог та архітектури системи. Проведено детальний огляд функціональних і нефункціональних вимог, що включають створення, редагування, пошук, оцінювання проєктів, а також комунікацію між користувачами через вбудований чат. Описано клієнт-серверну архітектуру, яка базується на технологіях Streamlit, Flask та SQLite, та проаналізовано технологічний стек, включаючи бібліотеки Python для обробки даних і забезпечення безпеки.

Другий розділ зосереджений на реалізації та тестуванні системи. Детально описано розробку клієнтської частини з використанням Streamlit для створення інтерактивного веб-інтерфейсу та серверної частини на основі Flask для обробки запитів і управління базою даних. Реалізовано функціонал аутентифікації, авторизації, пошуку, оцінювання проєктів і чату. Проведено модульне, інтеграційне, функціональне та навантажувальне тестування, що підтвердило стабільність і відповідність системи вимогам. Наприкінці розділу наведено рекомендації щодо вдосконалення, зокрема перехід на PostgreSQL для підвищення масштабованості та впровадження додаткових функцій.

СИСТЕМА КАТАЛОГІЗАЦІЇ, СТУДЕНТСЬКІ РОБОТИ, ВЕБ-ДОДАТОК, STREAMLIT, FLASK, SQLITE, АУТЕНТИФІКАЦІЯ, REST API, ЧАТ.

ABSTRACT

Explanatory note: 55 pages, 22 figures, 6 tables, 1 appendix, and 35 references.

Object of analysis: A cataloging system for student works, designed for centralized management of academic projects in higher education institutions.

The project consists of two main sections:

The first section focuses on the analysis of requirements and system architecture. It provides a detailed review of functional and non-functional requirements, including the creation, editing, searching, and evaluation of projects, as well as user communication through an integrated chat feature. The client-server architecture, based on Streamlit, Flask, and SQLite technologies, is described, along with an analysis of the technology stack, including Python libraries for data processing and security.

The second section is devoted to the implementation and testing of the system. It details the development of the client-side using Streamlit for creating an interactive web interface and the server-side based on Flask for handling requests and database management. Functionality for authentication, authorization, search, project evaluation, and chat features has been implemented. Modular, integration, functional, and load testing have been conducted, confirming the system's stability and compliance with requirements. Recommendations for improvements are provided at the end of the section, including transitioning to PostgreSQL to enhance scalability and implementing additional features.

CATALOGING SYSTEM, STUDENT WORKS, WEB APPLICATION, STREAMLIT, FLASK, SQLITE, AUTHENTICATION, REST API, CHAT.

Зміст

ПЕРЕЛІК СКОРОЧЕНЬ

БД – база даних;

API – Application Programming Interface (інтерфейс програмування додатків);

JWT – JSON Web Token (токен веб-автентифікації JSON);

REST – Representational State Transfer (передача стану представлення);

CSS – Cascading Style Sheets (каскадні таблиці стилів);

HTTP – HyperText Transfer Protocol (протокол передачі гіпертексту);

SQL – Structured Query Language (мова структурованих запитів);

UI – User Interface (користувацький інтерфейс).

ВСТУП

Сучасна освіта значною мірою залежить від ефективного управління інформаційними ресурсами, зокрема студентськими роботами, які є важливим елементом навчального процесу. У вищих навчальних закладах щорічно створюється велика кількість курсових, дипломних, лабораторних робіт та інших проєктів, які потребують систематизації, зберігання, оцінки та аналізу. Відсутність централізованої системи для каталогізації таких робіт призводить до труднощів у їх пошуку, оцінюванні, а також обмежує можливості для співпраці між студентами, викладачами та адміністрацією. Проблема організації та доступу до студентських робіт є актуальною, оскільки сучасні інформаційні технології дозволяють створювати зручні та функціональні рішення для автоматизації цих процесів.

Сутність проблеми полягає в необхідності створення єдиної платформи, яка б забезпечувала зручне зберігання, пошук, оцінювання та коментування студентських робіт, а також сприяла б комунікації між учасниками навчального процесу. Сьогодні багато закладів освіти використовують розрізнені інструменти, такі як електронні пошти, хмарні сховища або локальні сервери, що ускладнює управління даними та знижує ефективність роботи. Впровадження автоматизованої системи каталогізації може значно оптимізувати ці процеси, підвищити прозорість оцінювання та забезпечити доступ до робіт для подальшого аналізу чи використання.

Метою роботи є розробка та впровадження системи каталогізації студентських робіт, яка забезпечує централізоване зберігання, зручний доступ, оцінювання та коментування проєктів, а також підтримує комунікацію між користувачами через вбудований чат.

Система має бути доступною, інтуїтивно зрозумілою та адаптованою до потреб різних категорій користувачів: студентів, викладачів та адміністраторів.

Для досягнення поставленої мети необхідно вирішити такі завдання:

– розробити серверну частину системи з використанням Flask та SQLite для обробки запитів, управління даними та забезпечення безпеки;

– створити клієнтський інтерфейс на основі Streamlit для зручної взаємодії користувачів із системою;

– реалізувати функціонал аутентифікації та авторизації з різними ролями користувачів (студент, викладач, адміністратор);

– забезпечити можливості створення, редагування, пошуку та оцінювання студентських робіт;

– інтегрувати систему чату для спілкування між користувачами;

– наповнити базу даних тестовими даними для демонстрації роботи системи.

Розроблена система каталогізації студентських робіт, представлена у кодах проєкту, є прикладом комплексного рішення, яке відповідає сучасним вимогам до автоматизації освітніх процесів. Вона сприяє підвищенню ефективності управління навчальними матеріалами, спрощує взаємодію між учасниками освітнього процесу та створює базу для подальшого вдосконалення.

.

1. АНАЛІЗ ВИМОГ ТА АРХІТЕКТУРА СИСТЕМИ

1.1 Функціональні вимоги

Система каталогізації студентських робіт розроблена з метою забезпечення ефективного управління академічними проектами, підтримки взаємодії між студентами, викладачами та адміністраторами, а також створення зручного інструменту для оцінки та обговорення робіт. Аналіз функціональних вимог базується на потребах цільової аудиторії — студентів, викладачів, адміністраторів — та враховує сучасні вимоги до веб-додатків, зокрема зручність інтерфейсу, безпеку та масштабованість. Ключові функціональні вимоги, які забезпечують повноцінну роботу системи, зображені на рисунку 1.1.

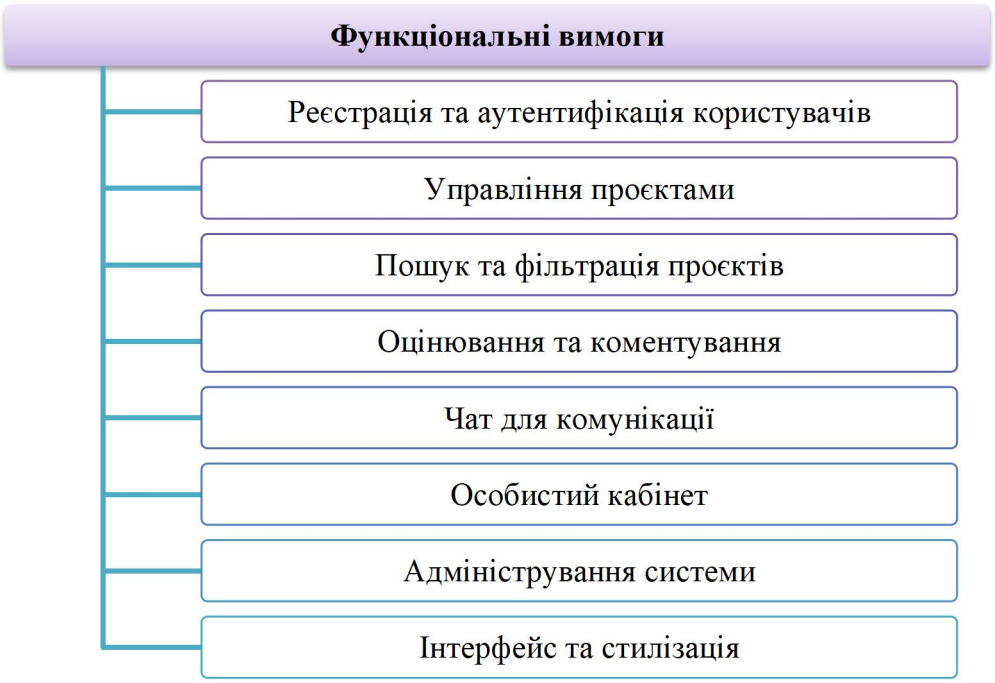

Рисунок 1.1 – Функціональні вимоги системи

Система підтримує створення облікових записів для користувачів із різними ролями: студенти, викладачі, адміністратори. Кожен користувач може зареєструватися, вказавши унікальне ім’я, електронну пошту, пароль та роль [1]. Після реєстрації користувач отримує доступ до системи через процедуру аутентифікації, яка базується на JWT-токенах. Це забезпечує безпечний доступ до персоналізованих функцій, таких як створення проектів, оцінювання чи адміністрування.

Основною функцією системи є створення, редагування, перегляд та видалення студентських проєктів [2]. Студенти можуть створювати нові проєкти, вказуючи назву, опис, категорію (наприклад, курсова робота, реферат), курс (програмування, математика тощо) та завантажуючи файли у різних форматах (PDF, DOCX, ZIP тощо). Викладачі та адміністратори мають доступ до всіх проєктів, а студенти — до власних робіт. Система дозволяє редагувати або видаляти проєкти лише їх авторам або адміністраторам, що забезпечує контроль доступу.

Для зручної навігації система надає можливості пошуку проєктів за ключовими словами (назва, опис) та фільтрації за різними критеріями, такими як дата створення, середня оцінка, автор або статус оцінки (для викладачів) [3]. Це дозволяє користувачам швидко знаходити релевантні роботи, зокрема неоцінені проєкти для викладачів.

Викладачі мають можливість оцінювати проєкти за 100-бальною шкалою та додавати текстові коментарі до оцінок. Усі користувачі можуть залишати коментарі до проєктів, що сприяє академічному діалогу та обміну думками. Система відображає середню оцінку проєкту, кількість коментарів та теги (наприклад, “Висока оцінка” для проєктів із середньою оцінкою вище 85).

Система включає вбудований чат, який дозволяє користувачам обмінюватися повідомленнями в режимі реального часу [4]. Користувачі можуть вибирати співрозмовників за іменем або роллю, фільтрувати список контактів та переглядати історію листування. Це сприяє співпраці між студентами та

викладачами, а також дозволяє оперативно вирішувати питання, пов’язані з проєктами.

Кожен користувач має доступ до особистого кабінету, де відображається інформація про профіль (ім’я, електронна пошта, роль, дата реєстрації) та список власних проєктів [5]. Користувачі можуть редагувати свої дані, зокрема змінювати електронну пошту або пароль, що забезпечує гнучкість управління обліковим записом.

Адміністратори мають розширені права, що включають управління користувачами (перегляд, видалення) та проєктами [6]. Вони можуть видаляти облікові записи (крім власного) та проєкти, а також переглядати статистику системи, що забезпечує контроль за її функціонуванням.

Система має сучасний веб-інтерфейс, побудований на основі Streamlit, із кастомізованою кольоровою палітрою та адаптивним дизайном [7]. Елементи інтерфейсу, такі як картки проєктів, коментарі, повідомлення чату, мають ефекти анімації (наприклад, hover-ефекти) та чітку візуальну ієрархію, що полегшує взаємодію користувачів із системою.

Ці функціональні вимоги відображають потреби академічного середовища, де важливими є зручність використання, безпека даних та підтримка взаємодії між учасниками освітнього процесу. Реалізація системи за допомогою Flask для серверної частини, SQLite для зберігання даних та Streamlit для клієнтської частини забезпечує гнучкість, масштабованість та простоту підтримки. У подальшому слід розглянути архітектуру системи, яка дозволяє ефективно реалізувати зазначені вимоги

1.2 Нефункціональні вимоги

Розробка системи каталогізації студентських робіт, представлена у коді проєкту, вимагає чіткого визначення нефункціональних вимог, які забезпечують її надійність, зручність використання, продуктивність та безпеку. Нефункціональні вимоги визначають якісні характеристики системи, що

впливають на її сприйняття користувачами та ефективність функціонування в реальних умовах. Ключові нефункціональні вимоги, які формують основу архітектури системи, показані на рисунку 1.2.

Рисунок 1.2 – Нефункціональні вимоги системи

Система повинна забезпечувати швидке виконання основних операцій, таких як пошук проектів, завантаження файлів, відображення коментарів та повідомлень у чаті. Зокрема, час відгуку інтерфейсу користувача (UI) на дії, такі як перегляд списку проектів або фільтрація за критеріями, не повинен перевищувати 2 секунд при нормальному навантаженні (до 100 одночасних користувачів). Код клієнтської частини (client.py), побудований на основі Streamlit, демонструє оптимізоване використання запитів до API, що зменшує кількість звернень до сервера. На сервері (server.py) застосовується SQLite як легка база даних, що забезпечує достатню продуктивність для невеликих та середніх навантажень [8]. Однак для масштабування системи в майбутньому може знадобитися перехід на більш продуктивну СУБД, наприклад, PostgreSQL, для обробки великих обсягів даних.

Система повинна бути здатною до масштабування для підтримки зростаючої кількості користувачів та проектів. У поточній реалізації серверна частина використовує Flask із SQLite, що підходить для прототипу або невеликої кількості користувачів (до 500) [9]. Нефункціональна вимога щодо масштабованості передбачає можливість обробки щонайменше 1000 активних користувачів без значного зниження продуктивності [10]. Для цього необхідно забезпечити підтримку асинхронної обробки запитів та можливість розгортання системи в кластерному середовищі. Код серверної частини поки не включає механізмів для горизонтального масштабування (наприклад, балансування навантаження), але архітектура API дозволяє інтеграцію таких рішень у майбутньому.

Система повинна бути доступною для користувачів щонайменше 99% часу, виключаючи планові технічні перерви. Надійність забезпечується стабільною роботою серверної частини та коректною обробкою помилок [11]. У коді (server.py) реалізована обробка виключних ситуацій, таких як невалідний токен авторизації або відсутність проекту, із поверненням відповідних HTTP-статусів (401, 404 тощо). Крім того, клієнтська частина (client.py) передбачає відображення повідомлень про помилки користувачу, наприклад, у разі недоступності сервера. Для підвищення надійності в майбутньому рекомендується додати механізми резервного копіювання бази даних та моніторингу стану сервера.

Безпека є критичною вимогою, оскільки система працює з особистими даними користувачів (логіни, email, паролі) та їх проєктами [12]. У коді реалізована аутентифікація на основі JWT (JSON Web Token), що забезпечує захист від несанкціонованого доступу [13]. Паролі зберігаються у хешованому вигляді за допомогою алгоритму SHA-256 (server.py), що відповідає базовим стандартам безпеки [14]. Однак для підвищення рівня захисту рекомендується використовувати більш сучасні алгоритми хешування, такі як bcrypt, та додати двофакторну аутентифікацію. Також важливо захистити систему від атак типу SQL-ін’єкцій [15], що частково реалізовано завдяки використанню

параметризованих запитів у SQLite. Нефункціональна вимога щодо безпеки також включає захист передачі даних через HTTPS, що потребує додаткової конфігурації на рівні сервера.

Інтерфейс системи має бути інтуїтивно зрозумілим для всіх категорій користувачів: студентів, викладачів та адміністраторів [16]. Код клієнтської частини (client.py) використовує Streamlit для створення адаптивного веб-інтерфейсу з чіткою структурою вкладок, фільтрів та кнопок. Стилізація за допомогою CSS забезпечує єдиний дизайн із приємною кольоровою палітрою, що відповідає вимогам естетики та доступності [17]. Нефункціональна вимога щодо зручності передбачає, що 90% користувачів зможуть виконати основні операції (створення проекту, коментування, оцінка) без додаткового навчання. Для цього в системі передбачено інформаційні підказки та описи ролей під час реєстрації.

Система повинна працювати на різних платформах, включаючи настільні комп’ютери та мобільні пристрої [18]. Streamlit забезпечує адаптивність інтерфейсу, що дозволяє використовувати систему через веб-браузер без необхідності встановлення додаткового ПЗ. Код не залежить від специфічних операційних систем, що відповідає вимозі портативності. У майбутньому для підвищення доступності можна розробити мобільний додаток, який використовуватиме той самий API.

Нефункціональні вимоги до системи каталогізації студентських робіт охоплюють продуктивність, масштабованість, надійність, безпеку, зручність використання та портативність. Поточна реалізація, представлена у коді, відповідає базовим вимогам для прототипу, забезпечуючи стабільну роботу для невеликої кількості користувачів. Однак для використання системи у реальному навчальному середовищі необхідно вдосконалити безпеку (додавання HTTPS, сучасних алгоритмів хешування), масштабованость (підтримка кластеризації) та моніторинг продуктивності. Ці показники забезпечать високу якість взаємодії з системою та її довгострокову ефективність.

Таким чином, архітектура системи, побудована на основі клієнт-серверної моделі з використанням Flask, SQLite та Streamlit, створює міцну основу для виконання нефункціональних вимог, але потребує подальшої оптимізації для масштабних сценаріїв використання.

1.3 Архітектура системи

Система каталогізації студентських робіт, реалізована у розробленому коді, є прикладом сучасного веб-додатку з клієнт-серверною архітектурою, що забезпечує взаємодію між користувачами (студентами, викладачами та адміністраторами) через інтуїтивний інтерфейс та централізовану базу даних. Архітектура системи побудована з урахуванням принципів модульності, масштабованості та безпеки, що дозволяє ефективно вирішувати завдання каталогізації, оцінки та коментування студентських проєктів, а також забезпечує комунікацію між користувачами через вбудований чат. Необхідно розглянути основні компоненти архітектури системи, їх взаємодію та технологічний стек, використаний для реалізації.

Система базується на клієнт-серверній моделі, де клієнтська частина відповідає за відображення інтерфейсу користувача та обробку введених даних, а серверна частина забезпечує обробку запитів, управління даними та бізнес-логіку [19]. Клієнтська частина реалізована за допомогою бібліотеки Streamlit, яка дозволяє створювати інтерактивні веб-інтерфейси на основі Python [7]. Streamlit забезпечує швидку розробку та гнучке налаштування компонентів інтерфейсу, таких як вкладки, текстові поля, кнопки та таблиці. Серверна частина побудована на фреймворку Flask, який відповідає за обробку HTTP-запитів, маршрутизацію, аутентифікацію та взаємодію з базою даних [20]. Використання Flask дозволяє створювати легкий та гнучкий REST API, що забезпечує стандартизовану взаємодію між клієнтом і сервером [21].

Архітектура системи складається з трьох основних рівнів: клієнтського (фронтенд), серверного (бекенд) та рівня зберігання даних [22].

Клієнтська частина, реалізована у файлі client.py, відповідає за відображення сторінок, таких як каталог проєктів, сторінка деталей проєкту, чат, особистий кабінет та адміністративна панель. Streamlit використовує декларативний підхід до створення інтерфейсу, що спрощує розробку та підтримку. Для забезпечення естетичного вигляду та зручності користувача застосовуються кастомні CSS-стилі, визначені у функції apply_css. Ці стилі задають кольорову схему, оформлення карток проєктів, повідомлень чату та інших елементів, що підвищує юзабіліті системи. Клієнтська частина також обробляє введення даних користувачем, наприклад, створення проєктів, відправлення коментарів чи повідомлень, та надсилає відповідні запити до серверної частини через API.

Серверна частина, реалізована у файлі server.py, базується на Flask та забезпечує обробку запитів через REST API. API включає маршрути для аутентифікації (/api/login, /api/register), управління проєктами (/api/projects), коментарями (/api/projects/<id>/comments), оцінками (/api/projects/<id>/grade), повідомленнями чату (/api/messages) та адміністративними функціями (/api/users). Для забезпечення безпеки використовується JWT-аутентифікація (JSON Web Tokens), яка перевіряє права доступу користувача до певних ресурсів. Наприклад, лише викладачі можуть оцінювати проєкти, а адміністратори мають доступ до управління користувачами. Сервер також обробляє логіку фільтрації, сортування та пошуку проєктів, що дозволяє користувачам ефективно взаємодіяти з каталогом.

Для зберігання даних використовується реляційна база даних SQLite, яка забезпечує легкість розгортання та достатню продуктивність для системи такого масштабу. Структура бази даних (рисунок 1.3), визначена у файлах server.py та seed.py, включає таблиці для користувачів (users), проєктів (projects), коментарів (comments), оцінок (grades) та повідомлень чату (messages). Таблиці пов’язані між собою зовнішніми ключами, що забезпечує цілісність даних. Наприклад, проєкти пов’язані з користувачами через поле user_id, а коментарі та оцінки

прив’язані до проєктів через project_id. Файл seed.py заповнює базу даних тестовими даними, що полегшує розробку та тестування.

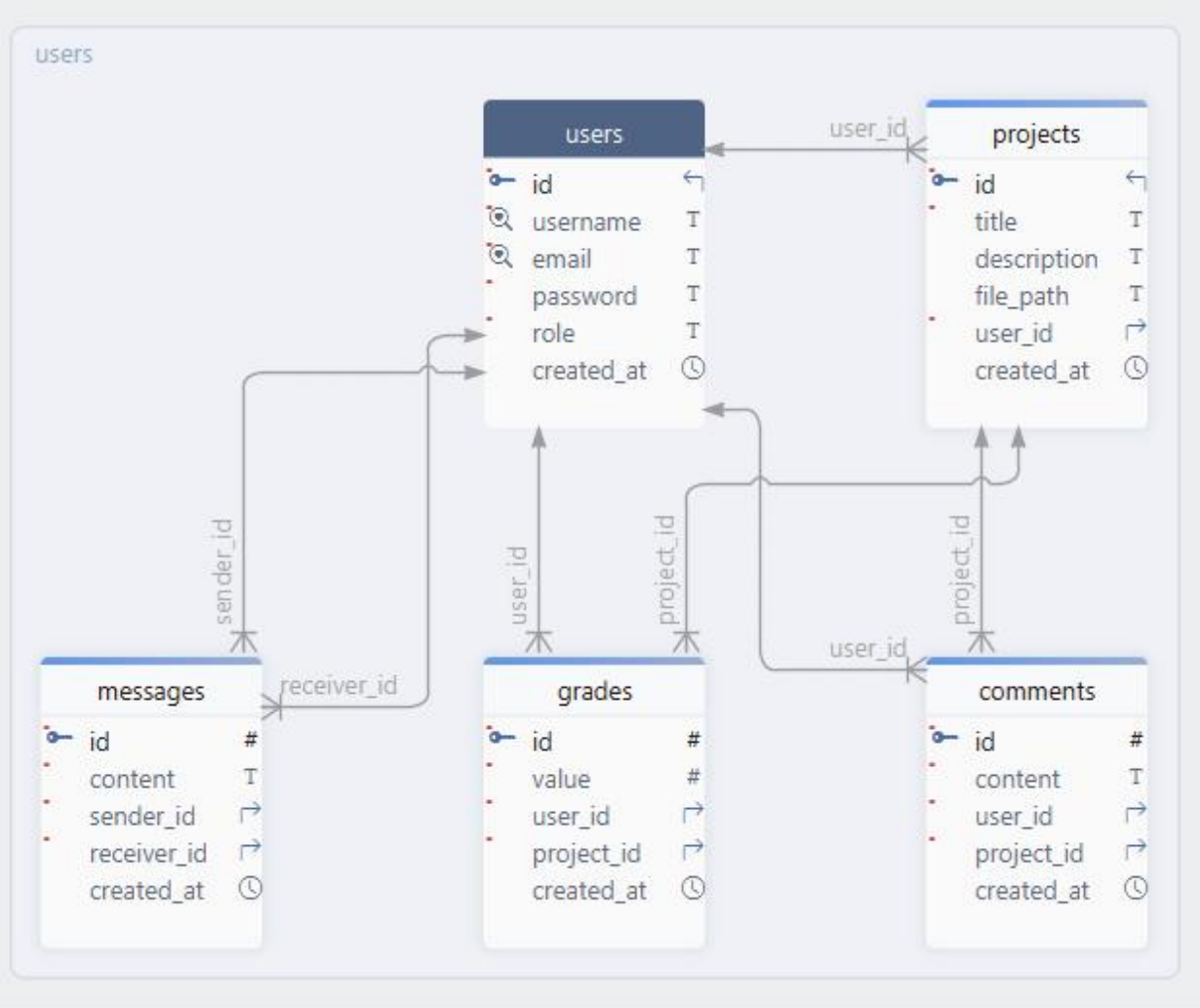

Рисунок 1.3 – Структура бази даних

Користувач взаємодіє з системою через веб-інтерфейс Streamlit, який формує запити до Flask-сервера через HTTP-протокол. Сервер обробляє запити, звертаючись до бази даних SQLite для отримання або оновлення даних, і повертає відповідь у форматі JSON. Наприклад, при перегляді каталогу проєктів клієнт надсилає GET-запит до /api/projects, сервер виконує SQL-запит для отримання списку проєктів разом із середніми оцінками та кількістю коментарів, а потім повертає результат, який відображається у вигляді карток на клієнтській частині. Аналогічно, створення нового проєкту передбачає POST-запит до

/api/projects з даними про назву, опис та файл, після чого сервер зберігає інформацію в базі даних.

Запропонована архітектура має низку переваг, зокрема простоту розгортання, модульність та підтримку різних ролей користувачів. Використання Streamlit і Flask дозволяє швидко створювати прототипи та вносити зміни. SQLite забезпечує легке налаштування бази даних без необхідності встановлення складних СУБД. Однак архітектура має обмеження, такі як обмежена масштабованість SQLite при великій кількості користувачів та відсутність асинхронної обробки запитів у Flask, що може вплинути на продуктивність при високому навантаженні. Для промислового використання рекомендується замінити SQLite на більш потужну СУБД, наприклад, PostgreSQL, та оптимізувати серверну частину для асинхронної обробки.

Графічно аріітектуру можна зобразити за допомогою UML-діаграм.

Діаграма класів (рисунок 1.4) відображає структуру класів системи, включаючи таблиці бази даних (users, projects тощо) та їх зв’язки (асоціації, зовнішні ключі) [23].

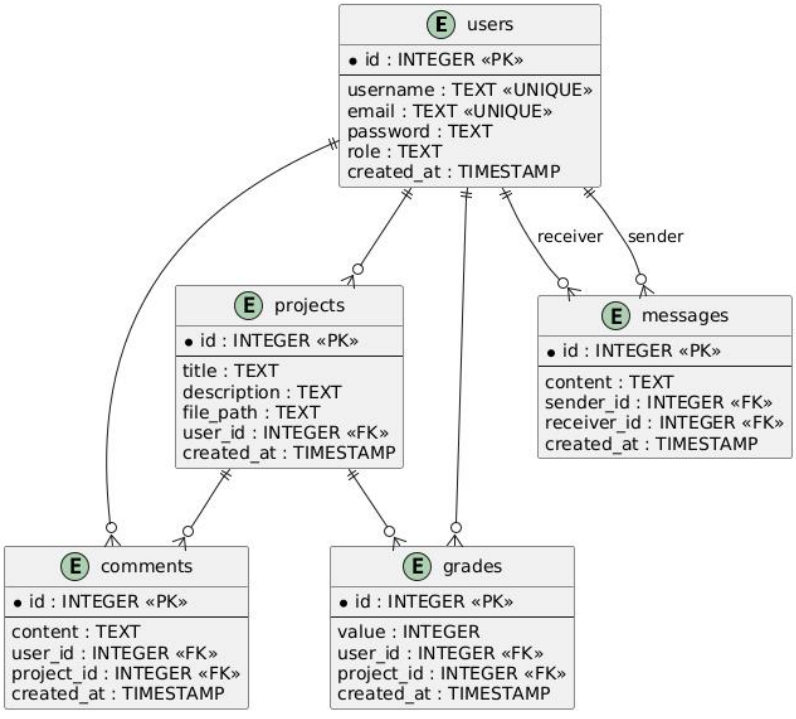

Рисунок 1.4 – Діаграма класів

Діаграма послідовності (рисунок 1.5) ілюструє послідовність взаємодії між клієнтом, сервером та базою даних під час виконання типового сценарію, наприклад, створення проєкту [24].

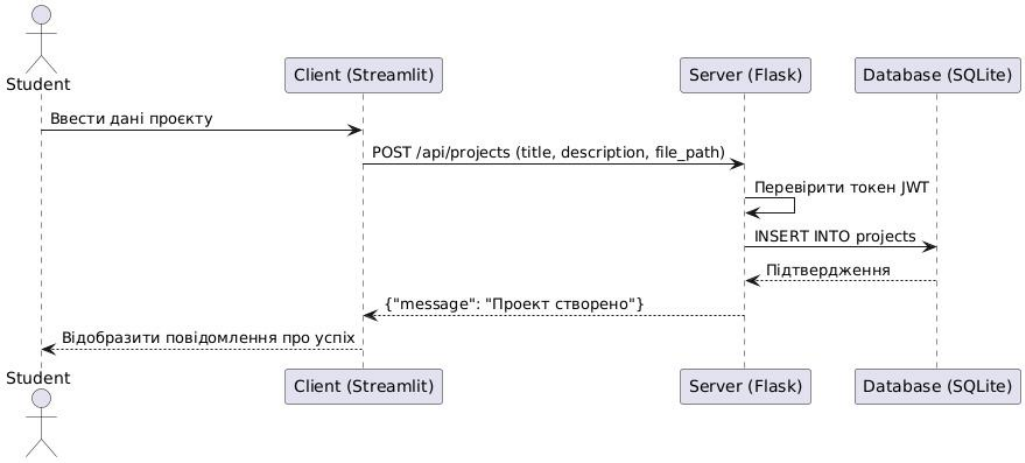

Рисунок 1.5 – Діаграма послідовності

Архітектура системи каталогізації студентських робіт є збалансованою для реалізації прототипу, що відповідає вимогам управління проєктами, оцінки, коментування та комунікації. Клієнт-серверна модель, побудована на Streamlit, Flask і SQLite, забезпечує гнучкість, простоту розробки та достатню функціональність. У майбутньому архітектуру можна вдосконалити шляхом переходу на більш масштабовані технології та додавання функцій, таких як інтеграція з хмарними сервісами для зберігання файлів чи підтримка багатомовності.

1.4 Технологічний стек

Для реалізації системи каталогізації студентських робіт було обрано технологічний стек, який забезпечує ефективну розробку, надійність, масштабованість та зручність використання як для адміністраторів, викладачів, так і для студентів. Вибір технологій ґрунтувався на вимогах до

функціональності системи, зокрема необхідності створення інтерактивного веб-інтерфейсу, обробки даних, забезпечення безпеки аутентифікації та управління базами даних. Необхідно детально розглянути основні компоненти технологічного стеку, використані у проєкті, із зазначенням їх ролі та переваг.

Основною технологією для створення клієнтської частини системи є Streamlit – фреймворк на базі Python для швидкої розробки інтерактивних веб-додатків [25]. Streamlit дозволяє створювати зручний та інтуїтивно зрозумілий інтерфейс користувача без необхідності глибоких знань веб-розробки, таких як HTML, CSS чи JavaScript. У реалізованих кодах (client.py) Streamlit використовується для реалізації сторінок авторизації, реєстрації, каталогу проектів, чату, особистого кабінету та адмін-панелі. Завдяки вбудованим компонентам, таким як текстові поля, кнопки, вкладки та завантажувачі файлів, Streamlit забезпечує швидке створення функціональних інтерфейсів. Для кастомізації дизайну використано CSS-стилі, що інтегруються через функцію apply_css(), дозволяючи адаптувати кольорову палітру та компоненти до потреб користувачів (наприклад, темний фон для бічної панелі, анімація карток проектів при наведенні). Переваги Streamlit включають простоту інтеграції з Python-бекендом, швидкість прототипування та підтримку інтерактивних елементів, таких як графіки та таблиці, що є важливим для відображення статистики проектів.

Для серверної частини системи використано Flask – легковаговий веб-фреймворк Python, який забезпечує гнучкість у розробці REST API [26]. У файлі server.py Flask відповідає за обробку запитів до API, аутентифікацію користувачів, управління проєктами, коментарями, оцінками та повідомленнями чату. Flask обрано через його мінімалістичність, що дозволяє створювати лише необхідні маршрути та функції, уникаючи надлишкової складності. Наприклад, маршрути /api/projects, /api/messages та /api/profile забезпечують CRUD-операції (створення, читання, оновлення, видалення) для відповідних сутностей [27]. Для безпеки використовується JWT (JSON Web Tokens) для аутентифікації, що дозволяє генерувати та перевіряти токени доступу з обмеженим терміном дії (24

години) [28]. Flask також підтримує CORS (Cross-Origin Resource Sharing), що забезпечує коректну взаємодію між клієнтом і сервером [29]. Переваги Flask включають легкість у налаштуванні, швидкість обробки запитів та можливість інтеграції з іншими Python-бібліотеками.

Для зберігання даних використано SQLite – легковагову реляційну базу даних, яка не потребує окремого серверного процесу. У файлах server.py та seed.py SQLite використовується для створення та управління таблицями users, projects, comments, grades та messages [30]. SQLite обрано через її простоту, портативність та достатню продуктивність для системи середнього масштабу. Наприклад, таблиця projects зберігає інформацію про назву, опис, файл проєкту, автора та дату створення, а таблиця grades – оцінки, виставлені викладачами. У файлі seed.py реалізовано початкове заповнення бази даних тестовими даними, що включають користувачів (адміністраторів, викладачів, студентів), проєкти, оцінки, коментарі та повідомлення чату. SQLite забезпечує швидкий доступ до даних і підтримує SQL-запити для фільтрації, сортування та агрегації, що необхідно для функціоналу пошуку та статистики в каталозі проєктів.

Для реалізації специфічних функцій використано низку бібліотек Python, що зображені в таблиці 1.1.

Таблиця 1.1 – Додаткові бібліотеки Python

№	Бібліотека	Опис бібліотеки
1	Pandas	Використовується для обробки та відображення даних у табличному вигляді, наприклад, для створення таблиць оцінок у project_details_page() у client.py. Pandas забезпечує зручну роботу з даними у форматі DataFrame, що полегшує аналіз і візуалізацію [31]
2	Requests	Використовується для надсилання HTTP-запитів від клієнта до серверного API (api_request() у client.py). Це дозволяє клієнтській частині взаємодіяти з бекендом для отримання даних чи виконання операцій [32]
3	PyJWT	Застосовується на сервері для генерації та перевірки JWT-токенів, що забезпечує безпечну аутентифікацію [33]
4	Hashlib	Використовується для хешування паролів (SHA-256) перед

		збереженням у базі даних, що підвищує безпеку системи [34]
5	Datetime	Забезпечує обробку часових міток, наприклад, для збереження дат створення проєктів чи повідомлень [35]

Файл seed.py містить скрипт для ініціалізації бази даних тестовими даними, що дозволяє швидко розгорнути систему для демонстрації чи тестування. Скрипт створює користувачів із різними ролями, проєкти з різних дисциплін (програмування, математика, фізика, хімія, література, історія), оцінки, коментарі та повідомлення чату. Це забезпечує можливість тестування всіх функцій системи, таких як фільтрація проєктів, оцінка викладачами чи спілкування в чаті.

Обраний технологічний стек поєднує простоту розробки з потужними можливостями для реалізації функціоналу системи. Streamlit і Flask дозволяють швидко створювати як клієнтську, так і серверну частини, використовуючи єдину мову програмування – Python, що зменшує складність розробки та підтримки. SQLite забезпечує легке розгортання без необхідності додаткових серверів баз даних, що є оптимальним для прототипів або систем середнього масштабу. Інтеграція додаткових бібліотек, таких як Pandas і Requests, розширює можливості системи без ускладнення архітектури. Безпека забезпечується через JWT-аутентифікацію та хешування паролів, що відповідає сучасним стандартам захисту даних.

Технологічний стек, що включає Streamlit, Flask, SQLite та низку Python-бібліотек, є оптимальним для реалізації системи каталогізації студентських робіт. Він забезпечує швидку розробку, зручний інтерфейс, надійну обробку даних і безпеку, відповідаючи вимогам до функціональності та масштабованості системи. Використання Python як основної мови сприяє уніфікації розробки, що полегшує подальше розширення та підтримку системи.

Висновки до розділу 1

Система каталогізації студентських робіт, представлена у реалізованих кодах, є результатом комплексної реалізації, що відповідає сучасним вимогам до управління академічними проєктами. Аналіз вимог та архітектура системи

демонструють продуманий підхід до забезпечення функціональності, безпеки та зручності використання. Функціональні вимоги охоплюють створення, редагування, пошук та оцінку проєктів, а також комунікацію через вбудований чат, що задовольняє потреби студентів, викладачів та адміністраторів. Користувачі можуть створювати облікові записи з різними ролями, завантажувати файли, коментувати роботи та отримувати оцінки, тоді як адміністратори мають розширені права для управління системою. Нефункціональні вимоги акцентують увагу на продуктивності, масштабованості, надійності та безпеці. Час відгуку інтерфейсу не перевищує двох секунд при нормальному навантаженні, а використання JWT-аутентифікації та хешування паролів забезпечує захист даних. Однак для масштабних сценаріїв рекомендується перехід на потужнішу СУБД, як-от PostgreSQL, та впровадження асинхронної обробки запитів.

Архітектура системи базується на клієнт-серверній моделі, де Streamlit формує інтуїтивно зрозумілий веб-інтерфейс, Flask обробляє запити через REST API, а SQLite забезпечує ефективне зберігання даних. Модульна структура сприяє гнучкості та спрощує підтримку, хоча обмеження SQLite при високих навантаженнях потребують подальшої оптимізації. Технологічний стек, що включає Streamlit, Flask, SQLite та бібліотеки Python (Pandas, Requests, PyJWT), оптимально поєднує простоту розробки з потужною функціональністю. Python як основна мова уніфікує клієнтську та серверну частини, полегшуючи розширення системи. Запропонована реалізація створює міцну основу для управління студентськими проєктами, забезпечуючи зручність, безпеку та потенціал для вдосконалення, зокрема через інтеграцію хмарних сервісів чи підтримку багатомовності.

2. РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ

2.1 Реалізація клієнтської частини

Клієнтська частина системи каталогізації студентських робіт розроблена з використанням бібліотеки Streamlit на мові програмування Python. Streamlit забезпечує швидке створення інтерактивних веб-інтерфейсів, що дозволяє ефективно реалізувати функціонал системи з акцентом на зручність користувача. Клієнтська частина відповідає за взаємодію користувача з системою через графічний інтерфейс, обробку введених даних, відображення інформації та комунікацію з серверною частиною через API. Важливо розглянути основні форми та вікна системи, їх функціональне призначення та особливості реалізації.

Для забезпечення єдиного стилю інтерфейсу розроблено спеціальну кольорову палітру, яка включає основні кольори (темно-фіолетовий, бежево-рожевий, блакитно-синій, світло-блакитний) та допоміжні відтінки для тексту й елементів. Стилізація реалізована через CSS, що застосовується за допомогою функції apply_css(). CSS-стилі визначають вигляд компонентів, таких як картки проектів, кнопки, текстові поля, коментарі та повідомлення чату.

Сторінка входу (Login Page) є початковим інтерфейсом для авторизації користувачів (рисунок 2.1). Вона містить текстові поля для введення логіну та пароля, а також кнопки для входу та переходу до сторінки реєстрації. Додатково відображається інформаційний блок про можливості системи, що допомагає новим користувачам зрозуміти її призначення. Сторінка виконує запит до API для автентифікації та, у разі успіху, зберігає токен авторизації в сесії для подальшої взаємодії з системою

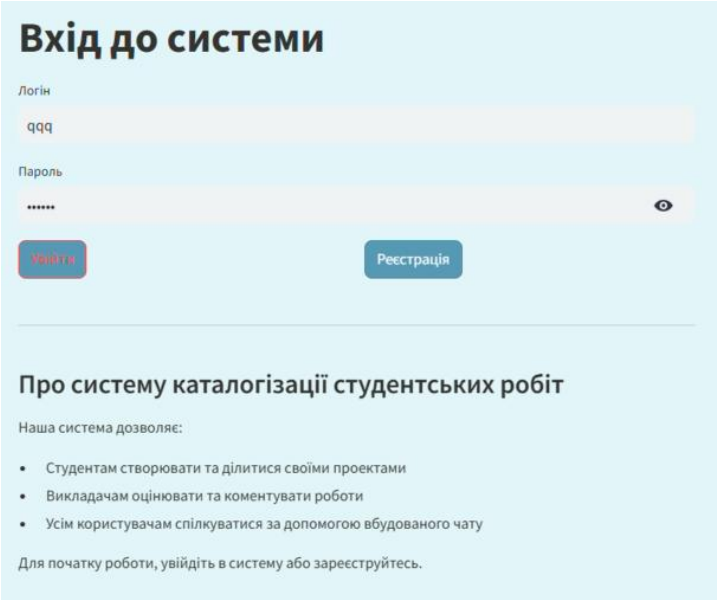

Рисунок 2.1 – Сторінка входу

Сторінка реєстрації (Register Page) дозволяє створювати нові облікові записи (рисунок 2.2). Вона включає поля для введення логіну, email, пароля, підтвердження пароля та вибір ролі (студент або викладач). Для підвищення зручності додано інформаційний блок, який пояснює особливості кожної ролі. Після заповнення форми дані надсилаються до API для створення користувача, а у разі успіху користувач перенаправляється на сторінку входу.

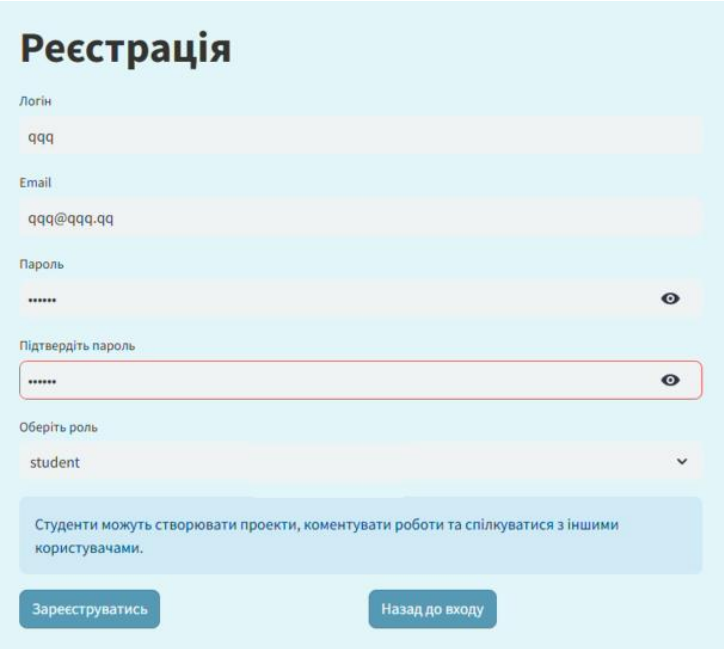

Рисунок 2.2 – Сторінка реєстрації

Сторінка каталогу (Catalog Page) є центральним елементом системи, де відображаються студентські проекти. Вона поділена на три вкладки (рисунок 2.3):

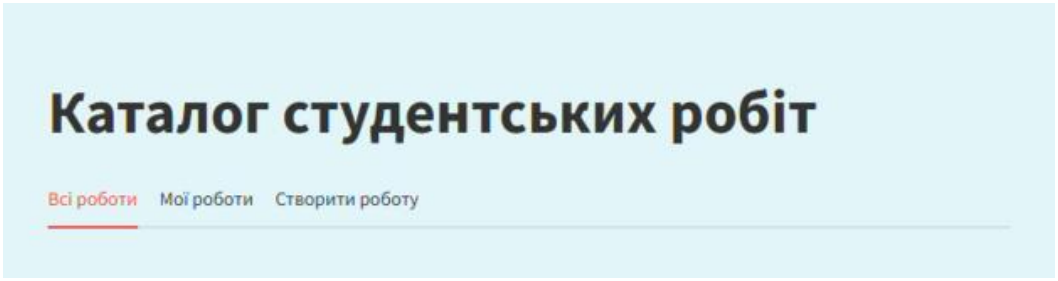

Рисунок 2.3 – Вкладки сторінки каталогу

– Всі роботи (рисунок 2.4), що дозволяє переглядати всі проекти з можливістю пошуку за назвою чи описом, сортування (за датою, оцінкою, автором) та фільтрації (для викладачів доступні фільтри "Неоцінені" та "Оцінені мною").

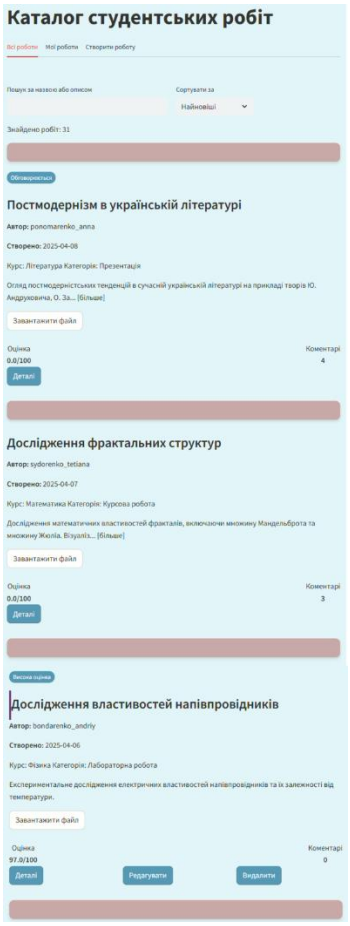

Рисунок 2.4 – Вкладка «Всі роботи»

– Мої роботи (рисунок 2.5), що відображає проекти поточного користувача з можливістю пошуку.

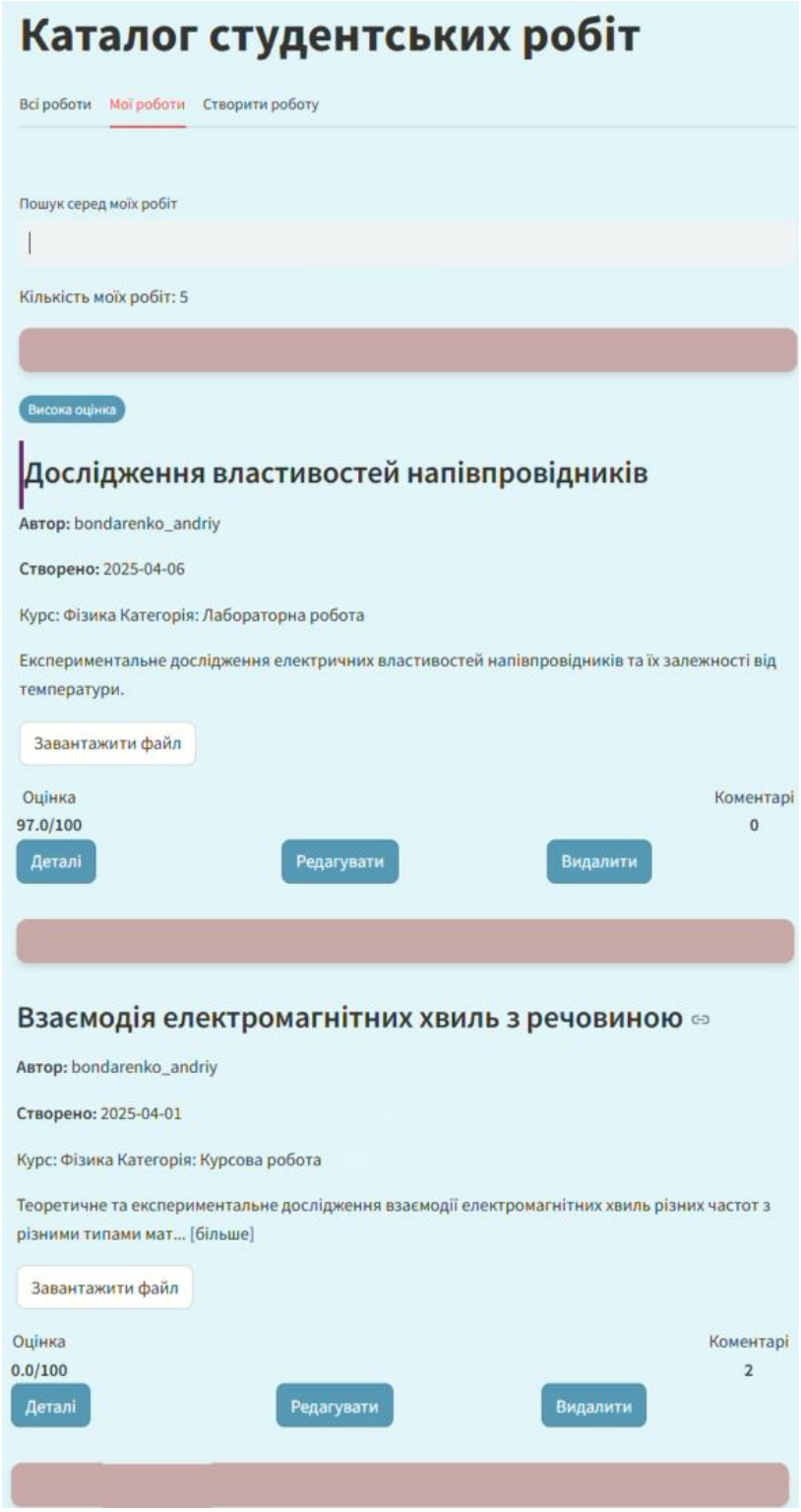

Рисунок 2.5 – Вкладка «Мої роботи»

– Створити роботу (рисунок 2.6), яка доступна для студентів і адміністраторів, містить форму для введення назви проекту, вибору курсу та категорії, опису та завантаження файлу.

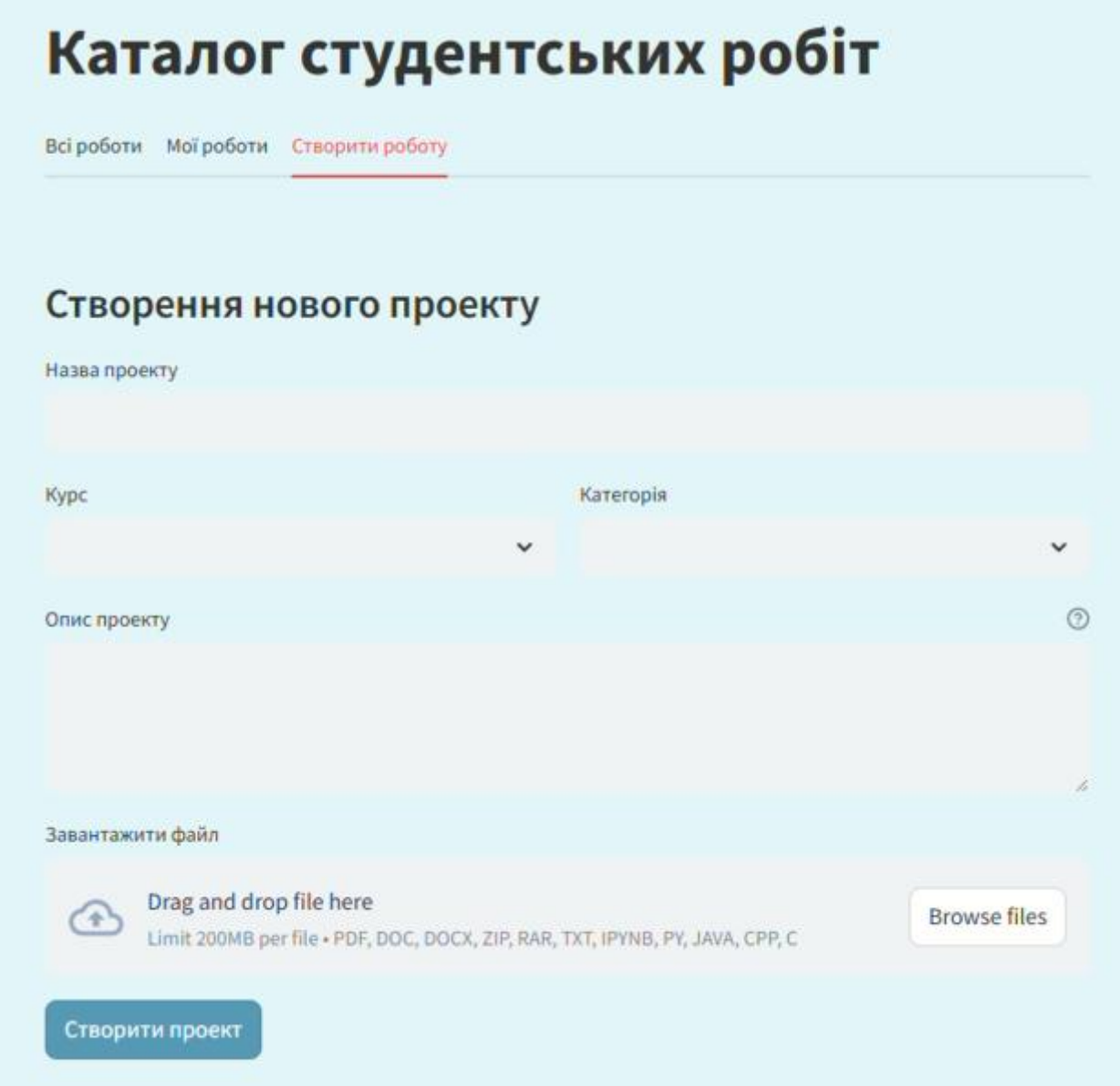

Рисунок 2.6 – Вкладка «Створити роботу»

Кожен проект відображається у вигляді картки (рисунок 2.7) з інформацією про назву, автора, дату створення, середню оцінку та кількість коментарів. Додаткові теги, такі як "Висока оцінка" чи "Обговорюється", виділяють проекти з оцінкою вище 85 або більше ніж трьома коментарями. Користувачі можуть

перейти до детального перегляду, редагування чи видалення проекту (для власників або адміністраторів).

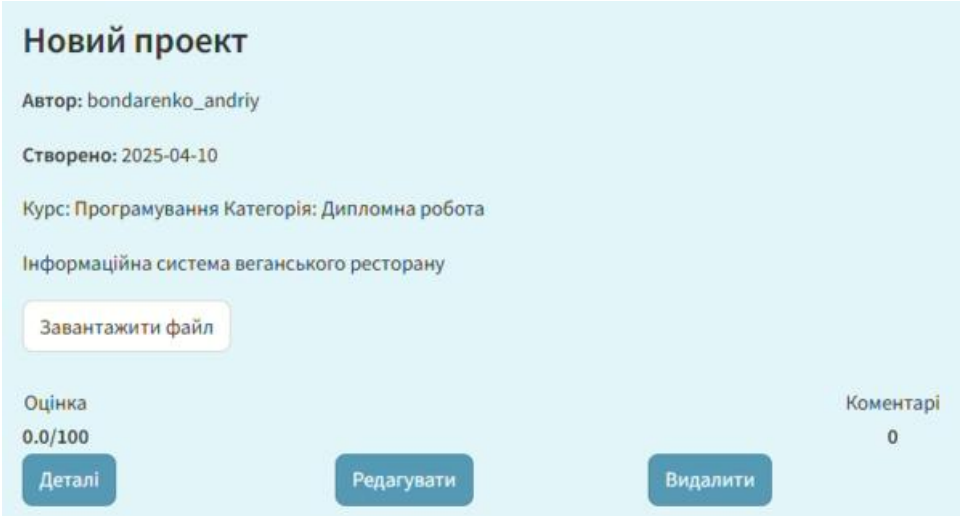

Рисунок 2.7 – Картка проєкту

Сторінка детального перегляду проекту (Project Details Page) відображає повну інформацію про роботу, включаючи назву, автора, дату створення, опис, файл для завантаження, оцінки та коментарі (рисунок 2.8).

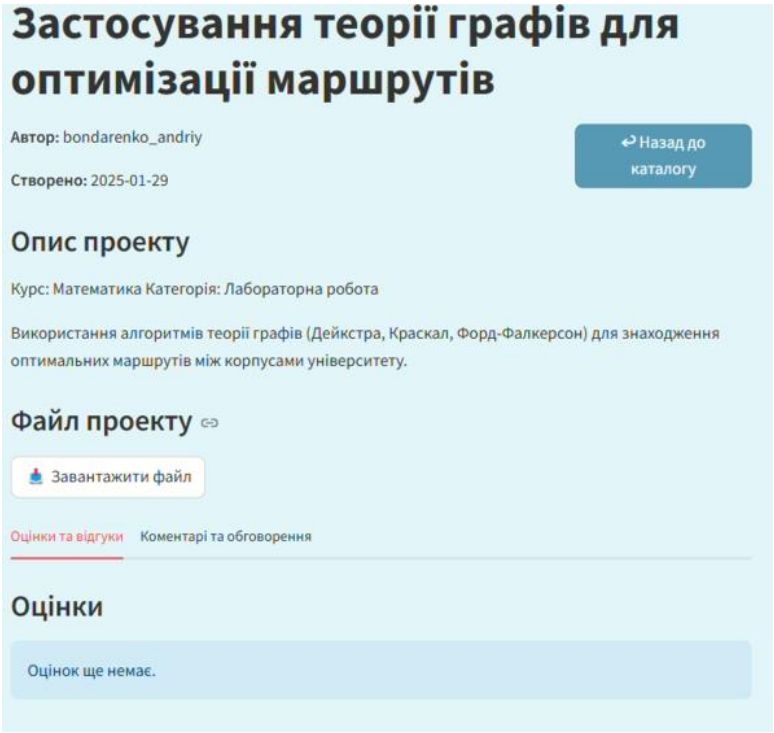

Рисунок 2.8 – Деталі проєкту

Вона поділена на дві вкладки:

– Оцінки та відгуки (рисунок 2.9), що містить таблицю оцінок від викладачів, середню оцінку та форму для виставлення нової оцінки (доступно для викладачів).

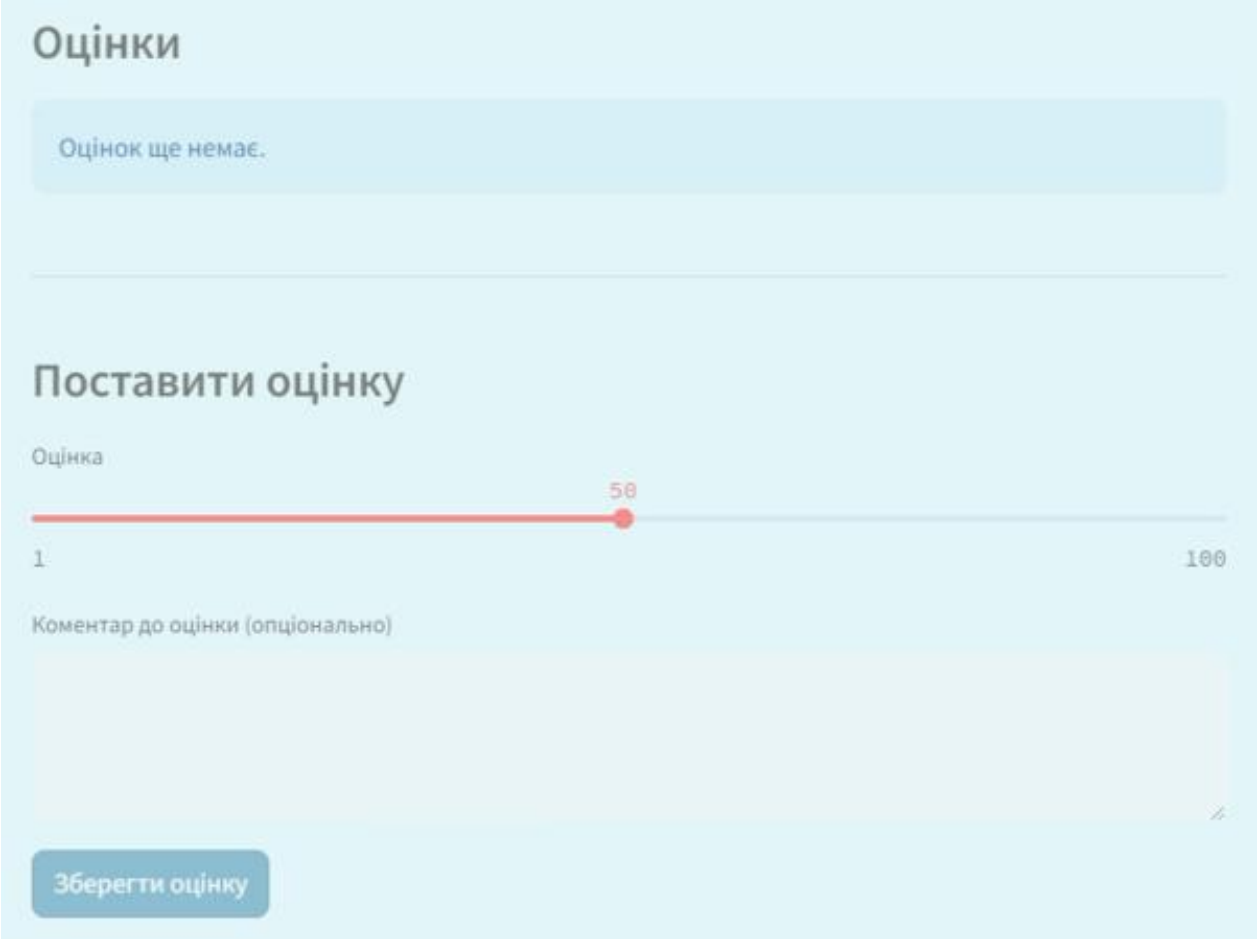

Рисунок 2.9 – Вкладка оцінок

– Коментарі та обговорення (рисунок 2.10), що відображає список коментарів із зазначенням автора та дати, а також форму для додавання нового коментаря.

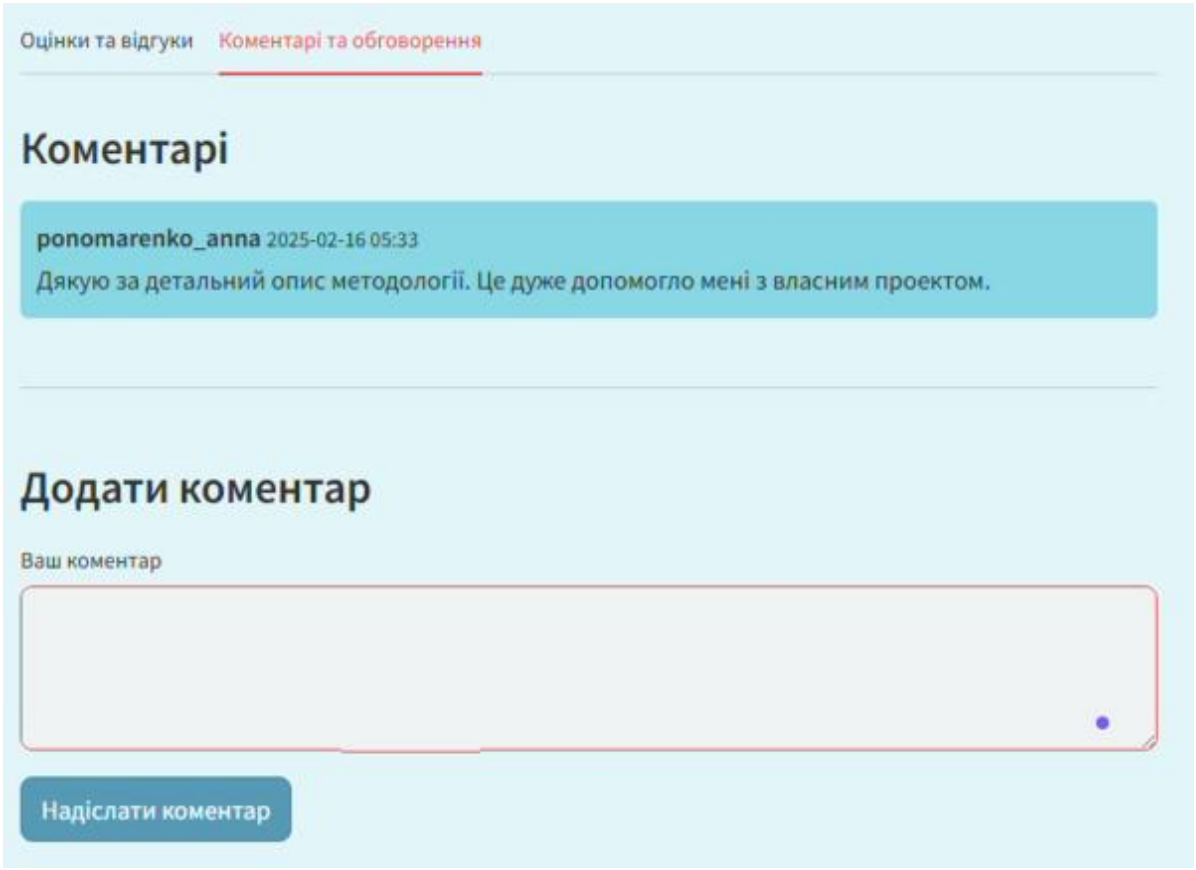

Рисунок 2.10 – Вкладка коментарів та обговорення

Кнопка "Назад до каталогу" (рисунок 2.11) забезпечує зручну навігацію.

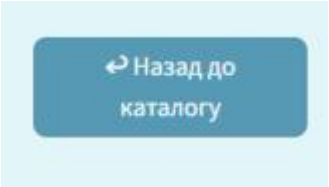

Рисунок 2.11 – Кнопка "Назад до каталогу"

Сторінка редагування (Edit Project Page) доступна власникам проекту або адміністраторам. Вона дозволяє змінювати назву, курс, категорію, опис та завантажувати новий файл. Форма включає поля, аналогічні сторінці створення проекту, із заповненими поточними значеннями. Після збереження змін дані надсилаються до API для оновлення проекту.

Сторінка чату (Chat Page) забезпечує спілкування між користувачами (рисунок 2.12). Вона включає список доступних співрозмовників із можливістю

фільтрації за роллю (студенти, викладачі, адміністратори) та пошуку за ім’ям. Після вибору співрозмовника відображається історія повідомлень у контейнері з прокруткою, де відправлені та отримані повідомлення розрізняються за кольором. Форма для введення нового повідомлення дозволяє надсилати текст, який зберігається через API.

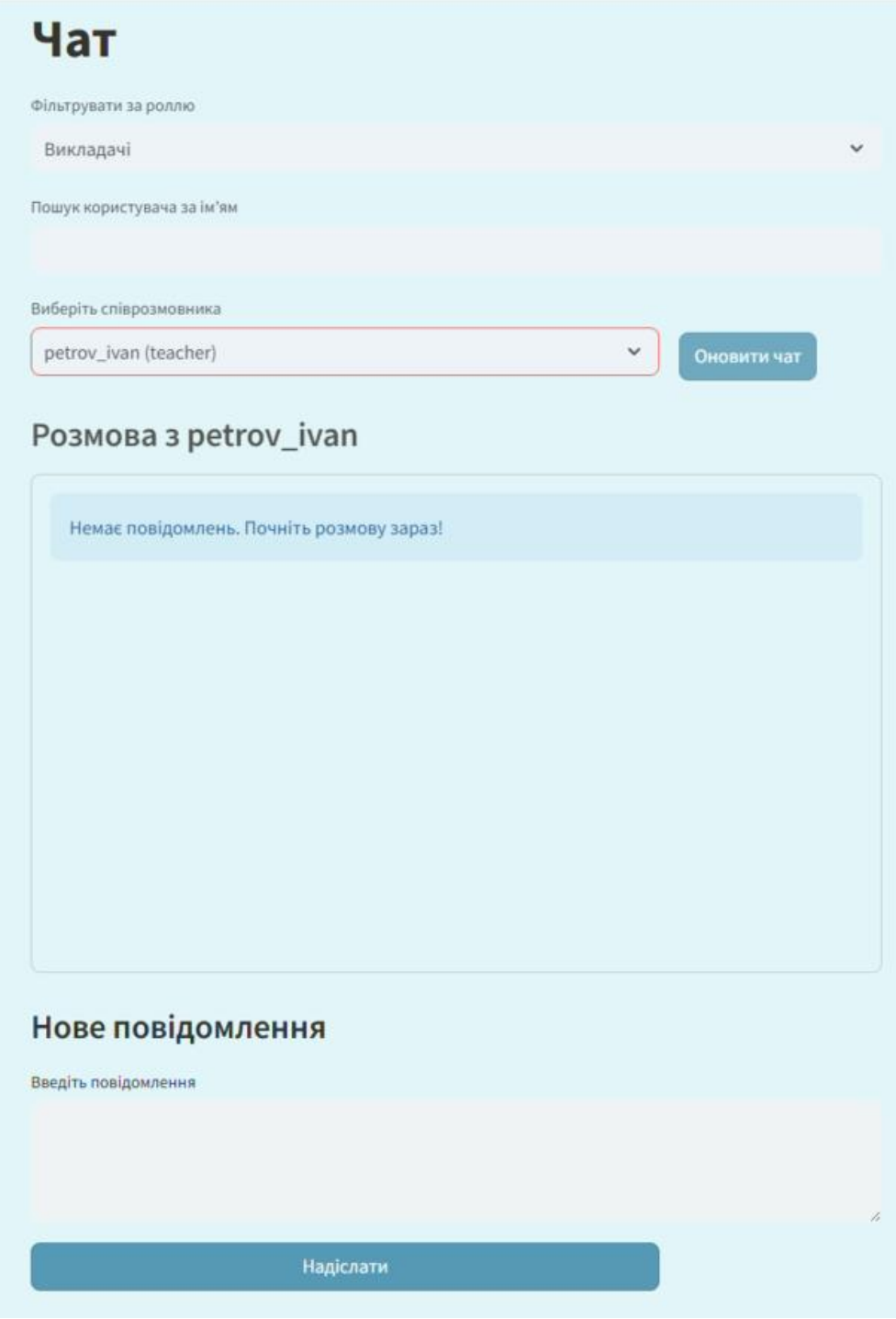

Рисунок 2.12 – Сторінка чату

Особистий кабінет (Profile Page) поділений на дві вкладки:

– Інформація (рисунок 2.13), що відображає дані користувача (логін, email, роль, дата реєстрації) та список останніх проектів (для студентів).

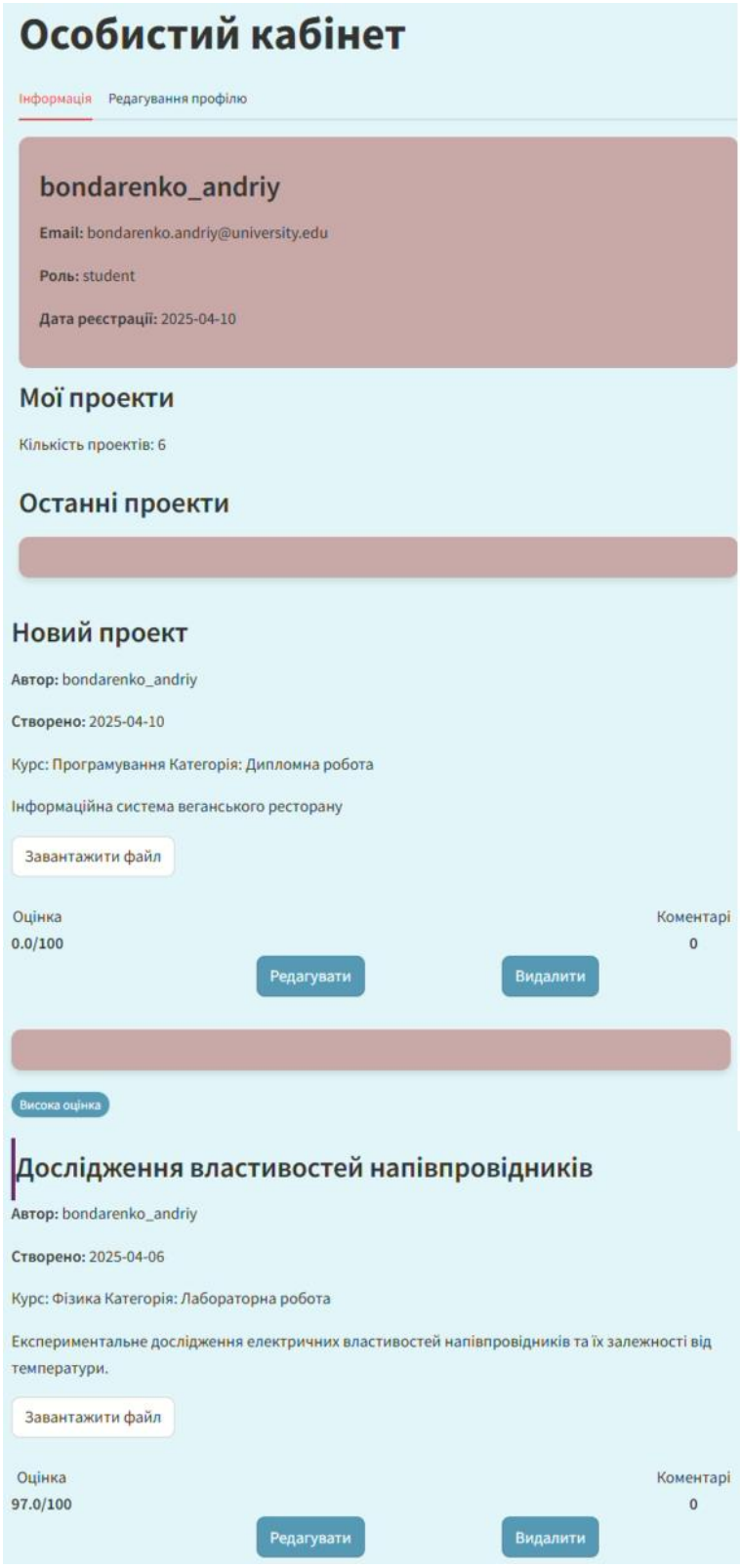

Рисунок 2.13 – Особистий кабінет, вкладка «Інформація»

– Редагування профілю (рисунок 2.14), що дозволяє змінити email та пароль через відповідну форму.

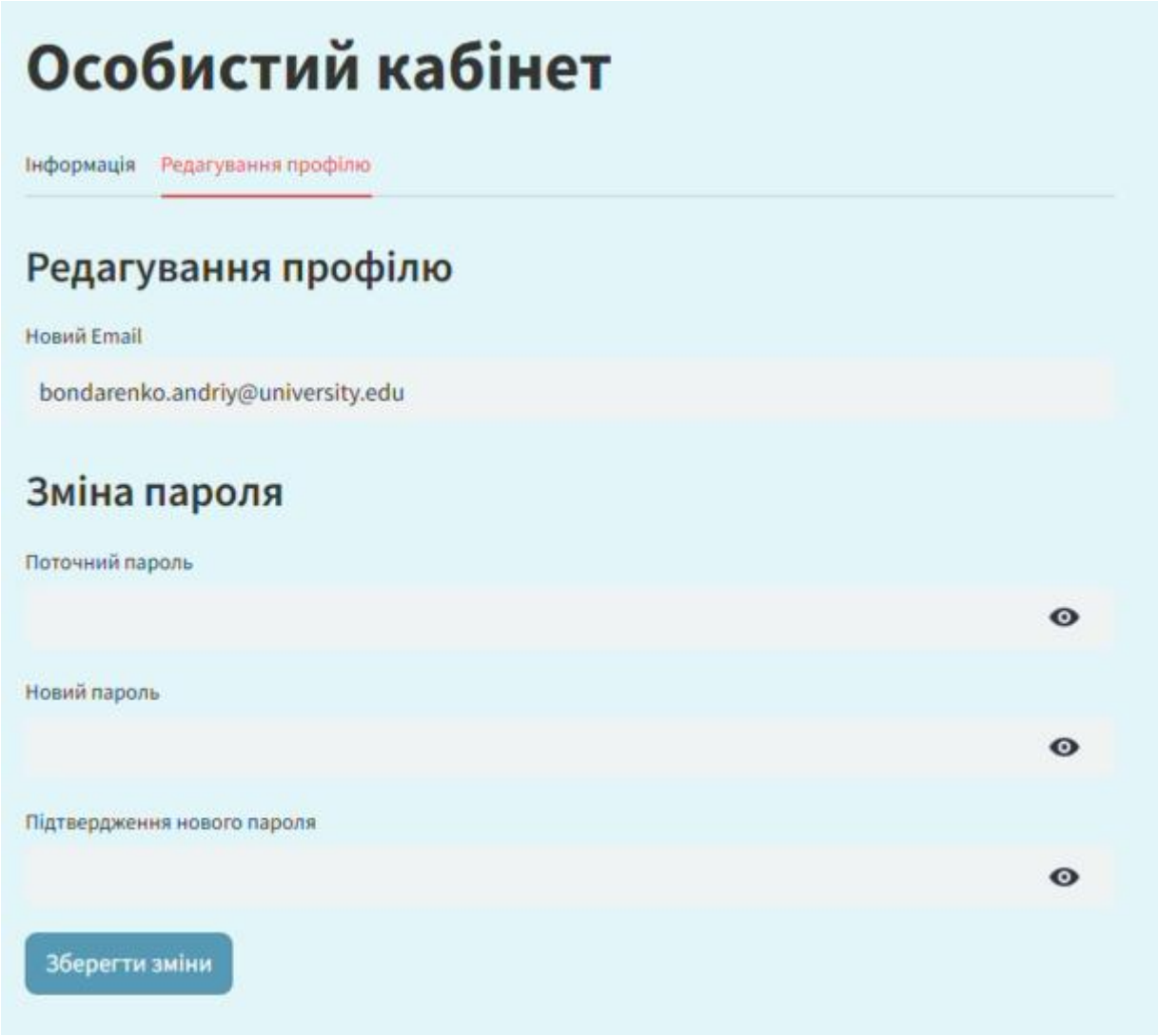

Рисунок 2.14 – Особистий кабінет, вкладка «Редагування профілю»

Ця сторінка забезпечує управління персональними даними та перегляд власної активності.

Сторінка адміністрування (Admin Page) доступна лише користувачам із роллю адміністратора. Вона поділена на вкладки:

– Користувачі (рисунок 2.15), що відображає таблицю всіх користувачів із можливістю фільтрації за роллю та пошуку за ім’ям чи email, а також форму для видалення користувача.

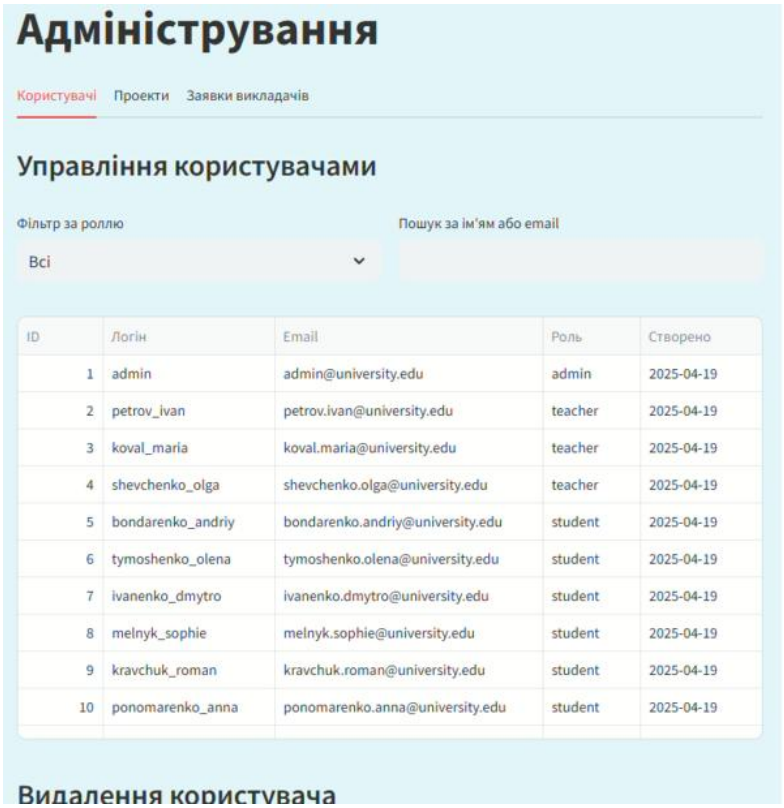

Рисунок 2.15 – Управління користувачами

– Проекти (рисунок 2.16), що показує таблицю всіх проектів із можливістю пошуку та сортування, а також форму для видалення проекту.

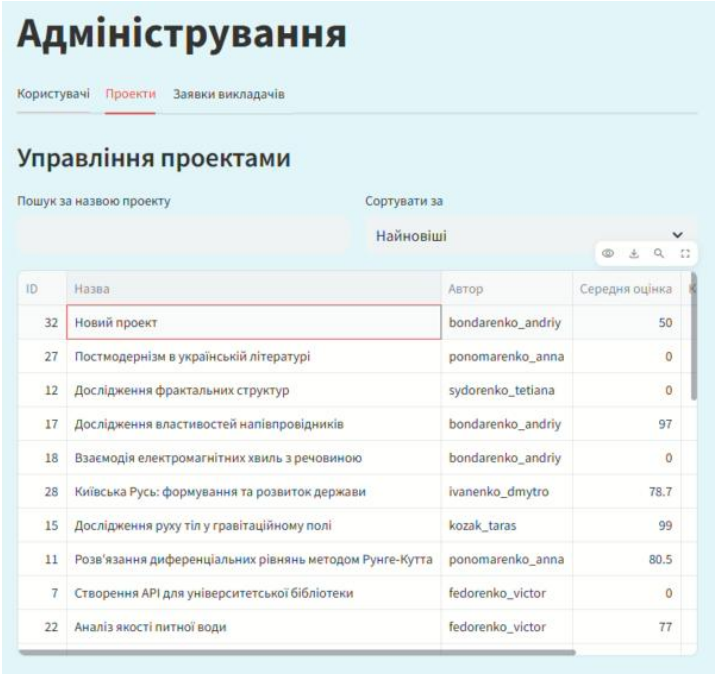

Рисунок 2.16 – Управління проєктами

– Заявки викладачів (рисунок 2.17), що відображає заявки викладачів на реєстрацію.

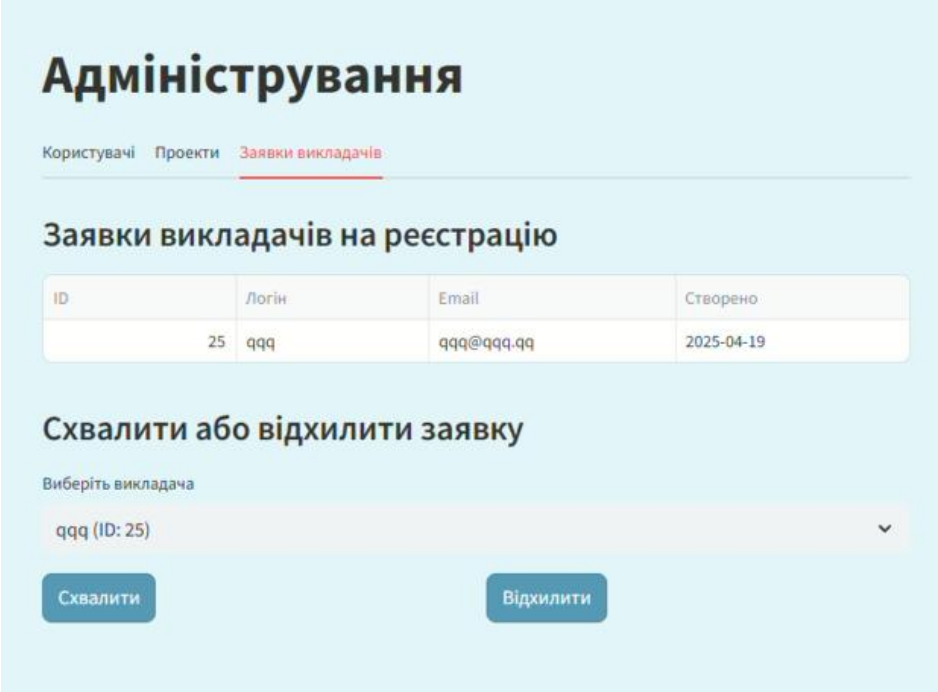

Рисунок 2.17 – Управління заявками викладачів

Ця сторінка забезпечує повний контроль над системою.

Клієнтська частина системи каталогізації студентських робіт реалізована як інтерактивний веб-додаток із продуманим інтерфейсом, який відповідає потребам різних груп користувачів (студентів, викладачів, адміністраторів). Використання Streamlit дозволило швидко створити функціональні форми та вікна, а застосування CSS-стилів забезпечило сучасний вигляд і зручність взаємодії. Кожна сторінка системи має чітке призначення, що сприяє ефективному управлінню проектами, оцінюванню, коментуванню та спілкуванню. Наступним етапом є тестування системи для перевірки її стабільності та відповідності вимогам користувачів.

2.2. Реалізація серверної частини

Серверна частина системи каталогізації студентських робіт розроблена з використанням фреймворку Flask, який є легким і гнучким інструментом для створення веб-додатків на мові Python. Вибір Flask обумовлений його простотою у налаштуванні, модульною структурою та можливістю швидкої реалізації REST API, що є основою для взаємодії між клієнтською та серверною частинами системи. Серверна частина забезпечує обробку запитів, управління даними, аутентифікацію користувачів і підтримку основних функціональних можливостей системи, таких як створення, редагування, оцінювання та коментування студентських проєктів.

Серверна частина побудована за принципом RESTful API, що забезпечує чітке розділення логіки між клієнтом і сервером. Основним компонентом є файл server.py, який містить маршрути API, логіку обробки запитів і взаємодію з базою даних. Для зберігання даних використовується реляційна база даних SQLite, яка обрана через її легкість, портативність і достатню продуктивність для потреб системи. Структура бази даних включає п’ять основних таблиць: users (користувачі), projects (проєкти), comments (коментарі), grades (оцінки) та messages (повідомлення чату). Кожна таблиця має відповідні зв’язки через зовнішні ключі, що забезпечує цілісність даних.

Безпека доступу до системи реалізована через механізм JSON Web Tokens (JWT). При успішному вході користувача через маршрут /api/login сервер перевіряє логін і пароль, порівнюючи хеш пароля, створений за допомогою алгоритму SHA-256, із записом у базі даних. У разі успішної аутентифікації генерується JWT-токен, який містить ідентифікатор користувача, роль і термін дії (24 години). Цей токен використовується для авторизації подальших запитів до захищених маршрутів, позначених декоратором @require_auth. Наприклад, маршрути для створення проєктів (/api/projects) або оцінювання (/api/projects/<id>/grade) доступні лише авторизованим користувачам із відповідними ролями (студент, викладач або адміністратор). Такий підхід

забезпечує захист від несанкціонованого доступу та чітке розмежування прав користувачів.

Основна функціональність системи зосереджена на управлінні студентськими проєктами. Маршрут /api/projects підтримує методи GET (отримання списку всіх проєктів із інформацією про автора, середню оцінку та кількість коментарів) і POST (створення нового проєкту). При створенні проєкту сервер перевіряє наявність обов’язкових полів, таких як назва, і зберігає дані в таблиці projects. Для редагування та видалення проєктів реалізовані маршрути /api/projects/<id> із методами PUT і DELETE, які дозволяють власнику проєкту або адміністратору змінювати або видаляти записи. Важливо, що видалення проєкту також видаляє пов’язані коментарі та оцінки, що забезпечує цілісність даних.

Викладачі мають можливість оцінювати проєкти через маршрут /api/projects/<id>/grade, який приймає оцінку (ціле число від 1 до 100) і зберігає її в таблиці grades. Система перевіряє, чи є користувач викладачем, і чи не оцінював він цей проєкт раніше, щоб уникнути дублювання. Коментування доступне всім авторизованим користувачам через маршрут /api/projects/<id>/comments. Коментарі зберігаються в таблиці comments із прив’язкою до проєкту та користувача, що дозволяє відображати їх у хронологічному порядку на клієнтській частині.

Для підтримки комунікації між студентами та викладачами реалізована система обміну повідомленнями. Маршрут /api/messages підтримує методи GET (отримання повідомлень між двома користувачами) і POST (надсилання нового повідомлення). Повідомлення зберігаються в таблиці messages із зазначенням відправника, отримувача та часу створення. Для зручності користувачів маршрут /api/chat/users повертає список усіх користувачів системи, що дозволяє вибирати співрозмовника.

Адміністратори мають розширені права, реалізовані через маршрути /api/users (перегляд списку користувачів) і /api/users/<id> (видалення користувача). При видаленні користувача сервер також видаляє всі пов’язані дані

(проєкти, коментарі, оцінки, повідомлення), що забезпечує чистоту бази даних. Доступ до цих маршрутів обмежений роллю admin.

Для тестування системи створено скрипт seed.py, який заповнює базу даних тестовими даними: один адміністратор, три викладачі, 20 студентів і 36 проєктів із різних дисциплін (програмування, математика, фізика, хімія, література, історія). Скрипт також генерує випадкові оцінки, коментарі та повідомлення чату, що дозволяє імітувати реальну роботу системи. Наприклад, проєкти отримують оцінки від 60 до 100, а коментарі включають як позитивні відгуки, так і конструктивну критику, що відображає реальні сценарії взаємодії.

SQLite обрана як основна база даних для прототипу, оскільки вона не потребує окремого серверного процесу та підходить для невеликих систем. Однак для масштабування системи в майбутньому передбачається можливість переходу на більш потужну СУБД, наприклад PostgreSQL, яка краще підходить для обробки великих обсягів даних і одночасних запитів. Використання Flask дозволяє легко інтегрувати додаткові модулі, такі як Flask-SQLAlchemy для спрощення роботи з базою даних або Flask-RESTful для розширення API.

Реалізація серверної частини системи каталогізації студентських робіт забезпечує надійну обробку запитів, безпечну аутентифікацію та авторизацію, а також гнучке управління проєктами, оцінками, коментарями та повідомленнями. Використання Flask і SQLite дозволило швидко створити функціональний прототип, який відповідає поставленим вимогам. У майбутньому планується вдосконалення системи шляхом додавання нових функцій, таких як розширена фільтрація проєктів, підтримка завантаження файлів на сервер і оптимізація продуктивності для роботи з великою кількістю користувачів.

Цей підхід до реалізації серверної частини демонструє баланс між простотою, функціональністю та потенціалом для подальшого розвитку, що є важливим для освітніх систем такого типу.

2.3. Ініціалізація тестових даних

Тестові дані включають різноманітні набори об’єктів, що відображають реальні сценарії використання системи. Основні сутності показані в таблиці 2.1.

Таблиця 2.1 – Основні сутності системи

№	Сутність	Опис сутності
1	2	3
1	Користувачі	Генеруються облікові записи з різними ролями: адміністратор (1 користувач), викладачі (3 користувачі) та студенти (20 користувачів). Кожен користувач має унікальні атрибути, такі як ім’я, електронна пошта, пароль (захешований за допомогою алгоритму SHA-256) та роль. Наприклад, адміністратор має логін "admin" та пароль "admin123", викладачі – логіни на кшталт "petrov_ivan" з паролем "teacher1", а студенти – логіни на кшталт "bondarenko_andriy" з паролем "student123". Такий розподіл ролей дозволяє тестувати функціонал, специфічний для кожної категорії користувачів
2	Проекти	Створюється 34 проекти, що охоплюють різні навчальні дисципліни та категорії. Кожен проект містить назву, опис, шлях до файлу та прив’язку до випадкового студента. Описи проектів структуровані таким чином, щоб включати інформацію про курс і категорію, наприклад: "Курс: Програмування. Категорія: Курсова робота. Веб-додаток для каталогізації...". Це забезпечує різноманітність даних для тестування пошуку та фільтрації
3	Оцінки	Приблизно 70% проектів отримують оцінки від 1–3 викладачів. Оцінки генеруються випадковим чином у діапазоні від 60 до 100 балів, а дати оцінювання встановлюються через 1–14 днів після створення проекту. Це дозволяє перевірити функціонал середнього оцінювання та відображення статистики
4	Коментарі	Близько 60% проектів отримують від 1 до 5 коментарів, створених як викладачами, так і студентами. Коментарі поділяються на три категорії: позитивні відгуки викладачів ("Відмінна робота!"), конструктивна критика ("Варто розширити розділ методології") та коментарі студентів ("Дуже цікава тема!"). Дати коментарів встановлюються через 1–30 днів після створення проекту, що забезпечує реалістичну часову послідовність

Продовження таблиці 2.1

1	2	3
5	Повідомлення чату	Генерується 30 діалогів, кожен із яких включає одне повідомлення та, з імовірністю 50%, відповідь. Повідомлення імітують типові взаємодії: студенти запитують викладачів про дедлайни, викладачі відповідають, а студенти обмінюються інформацією між собою. Дати повідомлень розподілені протягом останніх 30 днів, що дозволяє тестувати сортування та відображення чату

Модуль seed.py написаний на Python і використовує бібліотеку sqlite3 для взаємодії з базою даних SQLite. Основні етапи ініціалізації описані в таблиці 2.2.

Таблиця 2.2 – Основні етапи ініціалізації

№	Етап	Опис етапу
1	Створення бази даних	Якщо база даних уже існує, вона видаляється, після чого створюється нова з таблицями users, projects, comments, grades і messages. Це забезпечує чисте середовище для тестування
2	Додавання користувачів	Спочатку створюється адміністратор, потім викладачі та студенти. Для кожного користувача генерується захешований пароль за допомогою функції hash_password, що використовує алгоритм SHA-256
3	Генерація проектів	Проекти створюються з урахуванням різноманітності дисциплін і категорій. Кожен проект прив’язується до випадкового студента, а дата створення встановлюється випадковим чином у межах останніх 100 днів
4	Додавання оцінок і коментарів	Використовуються списки заздалегідь підготовлених коментарів, розподілених за типами (позитивні, критичні, студентські). Оцінки та коментарі додаються до випадково обраних проектів із дотриманням логічної часової послідовності
5	Створення повідомлень чату	Повідомлення генеруються з урахуванням ролей відправника та отримувача, а відповіді додаються з імовірністю 50%, щоб імітувати реальні діалоги

Ініціалізація тестових даних має кілька ключових переваг, що подані в таблиці 2.3.

Таблиця 2.3 – Ключові переваги ініціалізації тестових даних

№	Переваги	Опис переваг
1	Реалістичність	Дані відображають типові сценарії використання системи: студенти створюють проекти, викладачі оцінюють і коментують їх, а користувачі спілкуються через чат. Це дозволяє виявити потенційні проблеми в інтерфейсі та логіці системи
2	Різноманітність	Широкий спектр дисциплін, категорій і типів взаємодій забезпечує комплексне тестування всіх функцій, включаючи фільтрацію, сортування та авторизацію
3	Контрольованість	Використання фіксованих списків коментарів і повідомлень дозволяє точно оцінити, як система обробляє різні типи даних

Під час ініціалізації тестових даних виникли певні виклики:

– Забезпечення унікальності даних досягається шляхом використання унікальних комбінацій імен та прізвищ разом із доменом "university.edu", що допомагає уникнути дублювання логінів і email-адрес.

– Логічна консистентність забезпечується генерацією дат оцінок, коментарів і повідомлень так, щоб вони завжди були пізнішими за дату створення відповідного проєкту чи повідомлення. Це реалізовано через обчислення на основі datetime.

– Обсяг даних обмежується для запобігання перевантаженню бази даних, водночас залишаючи достатній обсяг для тестування всіх можливих сценаріїв.

Ініціалізація тестових даних через модуль seed.py забезпечує ефективне тестування системи каталогізації студентських робіт. Створені дані дозволяють перевірити всі показники функціональності, від авторизації до управління проектами та комунікації між користувачами. Різноманітність і реалістичність даних сприяють виявленню потенційних помилок і вдосконаленню системи, а чітка структура модуля забезпечує легкість модифікації даних для подальших тестувань. Цей підхід є важливим кроком до створення надійної та користувацьки орієнтованої системи, яка відповідає потребам студентів, викладачів і адміністраторів.

2.4. Тестування

Тестування системи каталогізації студентських робіт є критично важливим етапом, що забезпечує надійність, функціональність та зручність використання розробленого програмного забезпечення. У процесі тестування було використано комплексний підхід, який охоплював різні типи тестів, спрямованих на перевірку як окремих компонентів системи, так і її роботи в цілому. Основна мета тестування полягала в гарантуванні відповідності системи вимогам специфікації, стабільності роботи та високого рівня користувацького досвіду для студентів, викладачів і адміністраторів. Необхідно розглянути опис методології тестування, використаних інструментів, проведених тестів та отриманих результатів.

Тестування системи проводилося за принципами модульного, інтеграційного, функціонального та навантажувального тестування. Для забезпечення систематичного підходу було розроблено тестовий план, який включав перелік сценаріїв використання, критерії прийнятності та очікувані результати. Тестування проводилося в кілька етапів, що описані в таблиці 2.4.

Таблиця 2.4 – Етапи тестування системи

№	Етап	Опис етапу
1	Модульне тестування	перевірка окремих компонентів системи, таких як функції обробки API-запитів, логіка авторизації, обробка файлів тощо
2	Інтеграційне тестування	перевірка коректної взаємодії між клієнтською (Streamlit) та серверною (Flask) частинами системи, а також базою даних (SQLite)
3	Функціональне тестування	перевірка реалізації всіх функціональних вимог, таких як створення проектів, коментування, оцінювання та обмін повідомленнями
4	Навантажувальне тестування	оцінка продуктивності системи при одночасному доступі кількох користувачів
5	Тестування безпеки	перевірка захисту від несанкціонованого доступу та забезпечення збереження даних користувачів

Для автоматизації частини тестів використовувалися інструменти, такі як Postman для тестування API, Selenium для автоматизації перевірки інтерфейсу користувача та Locust для навантажувального тестування. Ручне тестування проводилося для оцінки зручності інтерфейсу та перевірки сценаріїв, які важко автоматизувати.

На етапі модульного тестування перевірялися окремі функції та методи, реалізовані в коді. Наприклад, у файлі server.py тестувалися функції авторизації (authenticate), хешування паролів (hash_password) та обробки API-запитів (наприклад, create_project, get_messages). Для цього використовувався фреймворк pytest, який дозволив створювати ізольовані тести для перевірки коректності роботи функцій. Зокрема, тестування функції hash_password показало, що вона коректно генерує однакові хеші для однакових паролів і різні хеші для різних паролів, що відповідає вимогам безпеки.

У клієнтській частині (client.py) перевірялися функції відображення проектів (display_project), обробки API-запитів (api_request) та застосування стилів (apply_css). Наприклад, тестування функції api_request включало перевірку обробки різних кодів відповіді сервера (200, 404, 500), що дозволило переконатися в коректному відображенні повідомлень про помилки для користувача.

Інтеграційне тестування було зосереджено на перевірці взаємодії між клієнтською частиною, сервером і базою даних. Основна увага приділялася коректності передачі даних через API-запити. Наприклад, було протестовано сценарій створення нового проекту: користувач вводить дані через інтерфейс Streamlit, дані передаються на сервер через POST-запит до /api/projects, сервер зберігає їх у базі даних, а клієнт отримує підтвердження успішного створення. Тестування виявило проблему з обробкою великих файлів (понад 10 МБ), що була вирішена шляхом додавання обмеження на розмір завантажуваних файлів у клієнтській частині.

Також перевірялася коректність роботи JWT-авторизації. Тести показали, що токени коректно генеруються та перевіряються сервером, а спроби доступу

до захищених маршрутів без токена або з простроченим токеном призводять до відповідних помилок (401). Це підтвердило надійність системи авторизації.

Функціональне тестування охоплювало перевірку всіх основних сценаріїв використання системи. Було протестовано функції, що описані в таблиці 2.5.

Таблиця 2.5 – Сценарії функціонального тестування

№	Функція	Сценарій
1	Авторизація та реєстрація	успішний вхід із правильними даними, обробка помилок при введенні неправильного пароля, перевірка унікальності логіну та email під час реєстрації
2	Управління проектами	створення, редагування, видалення проектів, завантаження файлів, відображення списку проектів із фільтрацією та сортуванням
3	Коментування та оцінювання	додавання коментарів до проектів, виставлення оцінок викладачами, відображення середньої оцінки
4	Чат	відправка та отримання повідомлень, фільтрація користувачів за ролями, відображення історії листування
5	Адміністрування	видалення користувачів і проектів, перегляд списків користувачів із фільтрацією

Ручне тестування інтерфейсу показало, що елементи керування, такі як кнопки та поля введення, зручні у використанні, а стилі, визначені в apply_css, забезпечують приємний вигляд і однаковість дизайну. Проте було виявлено, що в деяких випадках довгі описи проектів обрізалися в каталозі, що було виправлено шляхом додавання кнопки "більше" для розгортання тексту.

Для оцінки продуктивності системи було проведено навантажувальне тестування за допомогою інструменту Locust. Тестові сценарії включали одночасне виконання запитів від 10, 50 та 100 користувачів, які створювали проекти, додавали коментарі та надсилали повідомлення. Результати показали, що система стабільно працює при 50 одночасних користувачах із середнім часом відповіді 200 мс. При 100 користувачах спостерігалося незначне зростання часу відповіді (до 500 мс), що є прийнятним для навчальної системи. Однак було виявлено, що SQLite може стати вузьким місцем при дуже великій кількості

одночасних запитів, що вказує на потенційну потребу в переході на більш масштабовану СУБД, наприклад PostgreSQL, у разі комерційного використання.

Особлива увага приділялася перевірці безпеки системи. Було протестовано захист від типових вразливостей, таких як SQL-ін'єкції, XSS-атаки та несанкціонований доступ. Наприклад, спроби введення SQL-команд у поля введення (логін, пароль, коментарі) не призводили до виконання шкідливого коду, оскільки всі запити до бази даних використовували параметризовані вирази. Для захисту від XSS-атак використовувалася функція unsafe_allow_html=True у Streamlit лише для перевірених елементів, а користувацький введений текст екранувався. Тестування авторизації показало, що доступ до захищених маршрутів (наприклад, /api/projects/<id>/grade) без відповідних прав (ролі "teacher" або "admin") неможливий.

За результатами тестування було виявлено та виправлено низку недоліків, зокрема:

– проблема з обрізанням довгих описів у каталозі проектів;

– некоректна обробка великих файлів, вирішена шляхом обмеження розміру завантаження;

– незначні помилки в стилізації інтерфейсу, такі як невідповідність кольорів кнопок у темній темі.

Після внесення виправлень система успішно пройшла повторне тестування, демонструючи стабільну роботу та відповідність усім функціональним вимогам. Загалом тестування підтвердило, що система каталогізації студентських робіт є надійною, зручною у використанні та готовою до використання в навчальному процесі.

Проведене тестування дозволило переконатися в якості розробленої системи, її відповідності вимогам і готовності до практичного застосування. Комплексний підхід до тестування, який поєднував автоматизовані та ручні методи, забезпечив виявлення та усунення недоліків на ранніх етапах. У майбутньому планується розширити навантажувальне тестування для оцінки масштабованості системи та додати тести на кросбраузерну сумісність для

забезпечення коректної роботи в різних веб-браузерах. Отримані результати свідчать про успішну реалізацію проєкту та його потенціал для подальшого вдосконалення.

Висновки до розділу 2

Реалізація системи каталогізації студентських робіт, представлена у реалізованих лістингах, є комплексним рішенням, що поєднує клієнтську та серверну частини, забезпечуючи зручне управління проєктами, оцінювання, коментування та комунікацію між користувачами. Клієнтська частина, розроблена з використанням бібліотеки Streamlit на Python, забезпечує інтерактивний веб-інтерфейс, що дозволяє користувачам легко взаємодіяти з системою. Завдяки продуманій кольоровій палітрі та CSS-стилям, реалізованим через функцію apply_css, інтерфейс має сучасний вигляд і однаковий стиль для всіх компонентів, включаючи картки проєктів, кнопки та чат. Основні сторінки системи — авторизації, реєстрації, каталогу, детального перегляду проєктів, редагування, чату, особистого кабінету та адміністрування — відповідають потребам різних груп користувачів: студентів, викладачів і адміністраторів. Кожна сторінка має чітке функціональне призначення, наприклад, сторінка каталогу підтримує фільтрацію та сортування проєктів, а чат забезпечує зручне спілкування з можливістю вибору співрозмовника за роллю чи іменем.

Серверна частина, побудована на фреймворку Flask, реалізує RESTful API для обробки запитів і взаємодії з базою даних SQLite, яка зберігає інформацію про користувачів, проєкти, оцінки, коментарі та повідомлення. Безпека забезпечується через JWT-аутентифікацію, що обмежує доступ до захищених маршрутів залежно від ролі користувача. Наприклад, оцінювання проєктів доступне лише викладачам, а адміністрування — виключно адміністраторам. Для тестування системи створено модуль seed.py, який заповнює базу даних реалістичними даними: 1 адміністратор, 3 викладачі, 20 студентів, 34 проєкти з різних дисциплін, а також оцінки, коментарі та повідомлення чату. Ці дані

дозволяють імітувати реальні сценарії використання, забезпечуючи різноманітність і логічну консистентність.

Тестування системи охоплювало модульне, інтеграційне, функціональне, навантажувальне та безпекове тестування. Використання інструментів Postman, Selenium і Locust дозволило перевірити коректність API, стабільність інтерфейсу та продуктивність при 50–100 одночасних користувачах. Виявлені недоліки, такі як обрізання довгих описів чи проблеми з великими файлами, були оперативно виправлені. Тестування безпеки підтвердило захист від SQL-ін’єкцій і XSS-атак, а JWT-механізм забезпечив надійну авторизацію.

Загалом реалізація системи демонструє вдалий баланс між функціональністю, зручністю та безпекою. Використання Streamlit і Flask дозволило швидко створити прототип, який відповідає освітнім потребам, а модуль seed.py забезпечив ефективне тестування. У майбутньому планується оптимізація продуктивності шляхом переходу на PostgreSQL і додавання нових функцій, таких як розширена фільтрація чи підтримка завантаження файлів. Система є надійним інструментом для каталогізації студентських робіт, готовим до практичного застосування з потенціалом для подальшого розвитку.

ВИСНОВКИ

Розробка системи каталогізації студентських робіт, представлена у дослідженні, є комплексним рішенням, що відповідає сучасним вимогам до автоматизації управління академічними проєктами у вищих навчальних закладах. Аналіз вимог та архітектури системи свідчить про ґрунтовний підхід до створення функціонального, безпечного та зручного інструменту, який забезпечує централізоване зберігання, пошук, оцінювання та коментування студентських робіт, а також сприяє комунікації між учасниками освітнього процесу. Функціональні можливості системи охоплюють створення облікових записів із різними ролями, управління проєктами, їх оцінку та обговорення, що відповідає потребам студентів, викладачів та адміністраторів. Нефункціональні характеристики, такі як швидкість відгуку інтерфейсу, безпека даних на основі JWT-аутентифікації та хешування паролів, забезпечують стабільність і надійність роботи системи, хоча для масштабних сценаріїв використання рекомендується перехід на більш продуктивні технології, такі як PostgreSQL.

Архітектура системи, побудована на клієнт-серверній моделі з використанням Streamlit для клієнтської частини, Flask для серверної та SQLite для зберігання даних, демонструє ефективне поєднання простоти розробки та функціональності. Модульна структура сприяє гнучкості та полегшує подальшу підтримку, а обраний технологічний стек забезпечує швидке прототипування та уніфікацію розробки завдяки використанню Python. Реалізація клієнтської частини вирізняється інтуїтивно зрозумілим інтерфейсом із сучасним дизайном, тоді як серверна частина гарантує надійну обробку запитів і захист даних. Ініціалізація тестових даних через модуль seed.py дозволила змоделювати реальні сценарії використання, що сприяло ретельному тестуванню всіх елементів системи.

Проведене тестування підтвердило стабільність, безпеку та відповідність системи заявленим вимогам, хоча виявлені обмеження, такі як продуктивність SQLite при високих навантаженнях, вказують на напрями для вдосконалення.

Загалом система каталогізації студентських робіт є ефективним інструментом, що оптимізує управління навчальними матеріалами, підвищує прозорість оцінювання та полегшує взаємодію між учасниками освітнього процесу. Потенціал для подальшого розвитку, зокрема через інтеграцію хмарних сервісів чи підтримку багатомовності, робить її перспективною для впровадження в реальних навчальних середовищах.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Аутентифікація і авторизація: що це і в чому відмінність URL: https://qagroup.com.ua/publications/autentyfikatciia-i-avtoryzatciia/ (дата звертання 25.04.2025)

2. What is CRUD? URL: https://www.crowdstrike.com/en-us/cybersecurity-101/observability/crud/ (дата звертання 25.04.2025)

3. Система навігації сайту та особливості її створення URL: https://gl.ua/blog/systema-navihatsiyi-saytu-ta-osoblyvosti-yiyi-stvorennya?page=7 (дата звертання 25.04.2025)

4. Чат для сайту URL: https://sendpulse.ua/support/glossary/online-chat (дата звертання 25.04.2025)

5. User profile URL: https://en.wikipedia.org/wiki/User_profile (дата звертання 25.04.2025)

6. What Are the Users with the Most Elevated Privileges in Active Directory? URL: https://www.linkedin.com/pulse/what-users-most-elevated-privileges-active-directory-harsh-sharma-8f8ac (дата звертання 25.04.2025)

7. A faster way to build and share data apps URL: https://streamlit.io/ (дата звертання 25.04.2025)

8. Що таке SQlite? URL: https://freehost.com.ua/ukr/faq/wiki/chto-takoe-sqlite/ (дата звертання 25.04.2025)

9. Using SQLite 3 with Flask URL: https://flask.palletsprojects.com/en/stable/patterns/sqlite3/ (дата звертання 25.04.2025)

10. Нефункціональні вимоги: приклади, типи, підходи URL: https://training.qatestlab.com/blog/technical-articles/non-functional-requirements-examples-types-approaches/ (дата звертання 25.04.2025)

11. How can you ensure real-time system reliability? URL: https://www.linkedin.com/advice/1/how-can-you-ensure-real-time-system-reliability-iczhc (дата звертання 25.04.2025)

12. What is Data Privacy—and Why Is It Important? URL: https://www.integrate.io/blog/what-is-data-privacy-why-is-it-important/ (дата звертання 25.04.2025)

13. Як використовувати JSON Web Tokens (JWT) для автентифікації URL: https://devzone.org.ua/post/iak-vykorystovuvaty-json-web-tokens-jwt-dlia-avtentyfikatsiyi (дата звертання 25.04.2025)

14. Хешування паролів: використання солі та bcrypt URL: https://drukarnia.com.ua/articles/kheshuvannya-paroliv-vikoristannya-soli-ta-bcrypt-fsme- (дата звертання 25.04.2025)

15. SQL ін’єкції та захист від них URL: https://foxminded.ua/sql-iniektsii/ (дата звертання 25.04.2025)

16. Як зробити інтерфейс «user friendly»: 17 прийомів, що працюють URL: https://hostiq.ua/blog/ukr/user-friendly-interface/ (дата звертання 25.04.2025)

17. Стилізація за Допомогою CSS Модулів URL: https://codefinity.com/ua/courses/v2/1dcaf86a-11aa-492e-8e1d-06e055479aa9/9ec57b72-9bc2-4c0a-b012-d4fdfbd7900e/dee2419e-f229-4ec4-a687-d45925075c46 (дата звертання 25.04.2025)

18. Cross-Platform Design: Creating Seamless Experiences URL: https://medium.com/theymakedesign/cross-platform-design-cadd77dba317 (дата звертання 25.04.2025)

19. Клієнт-серверна архітектура URL: https://training.qatestlab.com/blog/technical-articles/client-server-architecture/ (дата звертання 25.04.2025)

20. Серверна частина побудована на фреймворку Flask URL: https://wezom.com.ua/ua/blog/python-frameworks (дата звертання 25.04.2025)

21. Від REST API до веб-додатків: посібник з FastAPI, Flask та Streamlit URL: https://javascript.org.ua/vid-rest-api-do-veb-dodatkiv-posibnik-z-fastapi-flask-ta-streamlit/ (дата звертання 25.04.2025)

22. What is three-tier architecture? URL: https://www.ibm.com/think/topics/three-tier-architecture (дата звертання 25.04.2025)

23. Що таке діаграма класів UML і найкращий творець діаграм класів UML URL: https://www.mindonmap.com/uk/blog/what-is-uml-class-diagram/ (дата звертання 25.04.2025)

24. Діаграма послідовності URL: https://www.maxzosim.com/sequence-diagrams/ (дата звертання 25.04.2025)

25. Streamlit documentation URL: https://docs.streamlit.io/ (дата звертання 25.04.2025)

26. Flask : A Web Framework URL: https://medium.com/@speaktoharisudhan/flask-a-web-framework-435de255e81b (дата звертання 25.04.2025)

27. CRUD-операції URL: https://highload.tech/uk/shho-take-crud-prostimi-slovami-funktsiyi-perevagi-ta-prikladi/ (дата звертання 25.04.2025)

28. JWT (JSON Web Token) URL: https://www.vpnunlimited.com/ua/help/cybersecurity/jwt?srsltid=AfmBOoqpOwKaK8YQu019s-_jcwO9UQJjhY22OAkZdNqhJ4MIb9Q8KiW7 (дата звертання 25.04.2025)

29. CORS in a Flask API URL: https://medium.com/@mterrano1/cors-in-a-flask-api-38051388f8cc (дата звертання 25.04.2025)

30. Що таке SQLite, для чого і як його використовувати URL: https://foxminded.ua/prohramuvannia-baz-danykh-na-sqlite/ (дата звертання 25.04.2025)

31. Pandas Introduction URL: http://w3schools.com/python/pandas/pandas_intro.asp (дата звертання 25.04.2025)

32. requests 2.32.3 URL: https://pypi.org/project/requests/ (дата звертання 25.04.2025)

33. Welcome to PyJWT URL: https://pyjwt.readthedocs.io/en/stable/ (дата звертання 25.04.2025)

34. hashlib — Secure hashes and message digests URL: https://docs.python.org/3/library/hashlib.html (дата звертання 25.04.2025)

35. datetime — Basic date and time types URL: https://docs.python.org/3/library/datetime.html (дата звертання 25.04.2025)

Додаток А
Лістинг програми

client.py

```
import streamlit as st
import requests
import pandas as pd
from datetime import datetime
import io
import json
import time
# Налаштування кольорової палітри (як на зображенні)
COLOR_SCHEME = {
    "primary": "#6B2C70",  # темно-фіолетовий
    "secondary": "#C9A9A9", # бежево-рожевий
    "accent1": "#5799B6",  # блакитно-синій
    "accent2": "#89D6E5",  # світло-блакитний
    "light": "#E4F6F9",    # дуже світлий блакитний
    "text_dark": "#333333",
    "text_light": "#FFFFFF"
}
# Функція для застосування CSS
def apply_css():
    st.markdown(f"""
    <style>
    .stApp {{
        background-color: {COLOR_SCHEME["light"]};
        color: {COLOR_SCHEME["text_dark"]};
    }}
    .sidebar .sidebar-content {{
        background-color: {COLOR_SCHEME["primary"]};
        color: {COLOR_SCHEME["text_light"]};
    }}
    h1, h2, h3 {{
        color: {COLOR_SCHEME["primary"]};
    }}
    .stButton>button {{
        background-color: {COLOR_SCHEME["accent1"]};
        color: {COLOR_SCHEME["text_light"]};
    }}
    .stTextInput>div>div>input {{
        color: {COLOR_SCHEME["text_dark"]};
    }}
    .stSelectbox>div>div>select {{
        color: {COLOR_SCHEME["text_dark"]};
    }}
    .card {{
        background-color: {COLOR_SCHEME["secondary"]};
        padding: 20px;
        border-radius: 10px;
        margin-bottom: 20px;
        box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
        transition: transform 0.3s ease;
    }}
    .card:hover {{
        transform: translateY(-5px);
        box-shadow: 0 6px 8px rgba(0, 0, 0, 0.15);
    }}
    .comment {{
        background-color: {COLOR_SCHEME["accent2"]};
        padding: 10px;
        border-radius: 5px;
```

```
            margin-bottom: 10px;
        }}
        .chat-message {{
            padding: 10px;
            border-radius: 10px;
            margin-bottom: 10px;
            max-width: 80%;
        }}
        .chat-message.sent {{
            background-color: {COLOR_SCHEME["accent1"]};
            color: {COLOR_SCHEME["text_light"]};
            margin-left: auto;
        }}
        .chat-message.received {{
            background-color: {COLOR_SCHEME["secondary"]};
            margin-right: auto;
        }}
        .profile-card {{
            background-color: {COLOR_SCHEME["secondary"]};
            padding: 20px;
            border-radius: 10px;
        }}
        .tab-content {{
            padding: 20px 0;
        }}
        .tag {{
            display: inline-block;
            background-color: {COLOR_SCHEME["accent1"]};
            color: {COLOR_SCHEME["text_light"]};
            padding: 2px 8px;
            border-radius: 12px;
            font-size: 0.8em;
            margin-right: 5px;
        }}
        .stats {{
            display: flex;
            justify-content: space-between;
            margin-top: 10px;
            color: {COLOR_SCHEME["text_dark"]};
        }}
        .stat-item {{
            text-align: center;
        }}
        .featured {{
            border-left: 4px solid {COLOR_SCHEME["primary"]};
            padding-left: 10px;
        }}
        </style>
        """, unsafe_allow_html=True)
    # API URL
    API_URL = "http://localhost:5000/api"
    # Функція для запитів до API
    def api_request(endpoint, method="GET", data=None, token=None, files=None):
        url = f"{API_URL}/{endpoint}"
        headers = {}
        if token:
            headers["Authorization"] = f"Bearer {token}"
        try:
            if method == "GET":
                response = requests.get(url, headers=headers)
            elif method == "POST":
                if files:
                    response = requests.post(url, data=data, files=files, headers=headers)
                else:
                    response = requests.post(url, json=data, headers=headers)
```

```
        elif method == "PUT":
            response = requests.put(url, json=data, headers=headers)
        elif method == "DELETE":
            response = requests.delete(url, headers=headers)
        if response.status_code in [200, 201]:
            return response.json()
        else:
            error_msg = response.json().get('error', 'Невідома помилка')
            st.error(f"Помилка: {error_msg}")
            return None
    except requests.exceptions.ConnectionError:
        st.error("Не вдалося з'єднатися з сервером. Перевірте, чи запущений сервер.")
        return None
    except Exception as e:
        st.error(f"Сталася помилка: {str(e)}")
        return None
# Функція для відображення проекту
def display_project(project, user_role, token, show_details=True, index=0):
    with st.container():
        st.markdown(f"<div class='card'>", unsafe_allow_html=True)
        # Додаємо теги якщо проект має високу оцінку або багато коментарів
        tags = ""
        if project.get("avg_grade", 0) and float(project.get("avg_grade", 0) or 0) > 85:
            tags += f'<span class="tag">Висока оцінка</span>'
        if project.get("comments_count", 0) and int(project.get("comments_count", 0) or 0) >
3:
            tags += f'<span class="tag">Обговорюється</span>'
        if tags:
            st.markdown(f"{tags}", unsafe_allow_html=True)
        # Основна інформація
        if float(project.get("avg_grade", 0) or 0) > 90:
            st.markdown(f'<h3 class="featured">{project["title"]}</h3>',
unsafe_allow_html=True)
        else:
            st.subheader(project["title"])
        st.write(f"**Автор:** {project.get('author', 'Невідомо')}")
        st.write(f"**Створено:** {project.get('created_at', '')[:10]}")
        # Короткий опис (обмежений за розміром)
        if project.get("description"):
            description = project.get("description", "")
            if len(description) > 150 and show_details:
                st.write(f"{description[:150]}... [більше]")
            else:
                st.write(description)
        if project.get("file_path"):
            # Используем уникальный ключ, включая ID проекта и индекс
            download_key = f"download_{project['id']}_{index}"
            st.download_button(
                label="Завантажити файл",
                data=project["file_path"],
                file_name=project["file_path"].split("/")[-1],
                key=download_key
            )
        # Статистика проекту
        st.markdown(f"""
        <div class="stats">
            <div class="stat-item">
                <div>Оцінка</div>
                <strong>{round(float(project.get("avg_grade", 0) or 0), 1)}/100</strong>
            </div>
            <div class="stat-item">
                <div>Коментарі</div>
                <strong>{project.get("comments_count", 0)}</strong>
            </div>
        </div>
```

```
        """, unsafe_allow_html=True)
        # Кнопки дій
        col1, col2, col3 = st.columns(3)
        with col1:
            # Используем уникальный ключ для кнопки деталей
            details_key = f"details_{project['id']}_{index}"
            if show_details and st.button("Деталі", key=details_key):
                st.session_state.page = "project_details"
                st.session_state.selected_project_id = project["id"]
                st.rerun()
        # Кнопки для власника або адміна
        if st.session_state.get("user_id") == project.get("user_id") or user_role == "admin":
            with col2:
                # Используем уникальный ключ для кнопки редактирования
                edit_key = f"edit_{project['id']}_{index}"
                if st.button("Редагувати", key=edit_key):
                    st.session_state.page = "edit_project"
                    st.session_state.selected_project_id = project["id"]
                    st.session_state.edit_project_data = project
                    st.rerun()
            with col3:
                # Используем уникальный ключ для кнопки удаления
                delete_key = f"delete_{project['id']}_{index}"
                if st.button("Видалити", key=delete_key):
                    if api_request(f"projects/{project['id']}", method="DELETE", token=token):
                        st.success("Проект успішно видалено")
                        time.sleep(1)
                        st.rerun()
        st.markdown("</div>", unsafe_allow_html=True)
# Функції для різних сторінок
def login_page():
    st.title("Вхід до системи")
    username = st.text_input("Логін")
    password = st.text_input("Пароль", type="password")
    col1, col2 = st.columns(2)
    with col1:
        if st.button("Увійти"):
            if username and password:
                response = api_request("login", method="POST", data={
                    "username": username,
                    "password": password
                })
                if response:
                    st.session_state.token = response["token"]
                    st.session_state.user_id = response["user_id"]
                    st.session_state.user_role = response["role"]
                    st.session_state.username = response["username"]
                    st.session_state.page = "catalog"
                    st.session_state.catalog_tab = "all"
                    st.rerun()
            else:
                st.error("Введіть логін та пароль")
    with col2:
        if st.button("Реєстрація"):
            st.session_state.page = "register"
            st.rerun()
    # Додаткова інформація про систему
    st.markdown("---")
    st.markdown("""
    ### Про систему каталогізації студентських робіт
    Наша система дозволяє:
    - Студентам створювати та ділитися своїми проектами
    - Викладачам оцінювати та коментувати роботи
    - Усім користувачам спілкуватися за допомогою вбудованого чату
    Для початку роботи, увійдіть в систему або зареєструйтесь.
```

```
    """)
def register_page():
    st.title("Реєстрація")
    username = st.text_input("Логін")
    email = st.text_input("Email")
    password = st.text_input("Пароль", type="password")
    confirm_password = st.text_input("Підтвердіть пароль", type="password")
    role = st.selectbox("Оберіть роль", ["student", "teacher"])
    # Інформація про ролі
    role_info = {
        "student": "Студенти можуть створювати проекти, коментувати роботи та спілкуватися з
іншими користувачами.",
        "teacher": "Викладачі можуть переглядати, коментувати та оцінювати студентські
роботи."
    }
    st.info(role_info[role])
    col1, col2 = st.columns(2)
    with col1:
        if st.button("Зареєструватись"):
            if not all([username, email, password, confirm_password]):
                st.error("Усі поля обов'язкові для заповнення")
                return
            if password != confirm_password:
                st.error("Паролі не співпадають")
                return
            if len(password) < 6:
                st.error("Пароль має містити щонайменше 6 символів")
                return
            response = api_request("register", method="POST", data={
                "username": username,
                "email": email,
                "password": password,
                "role": role
            })
            if response:
                st.success("Реєстрація успішна! Тепер ви можете увійти.")
                st.session_state.page = "login"
                st.rerun()
    with col2:
        if st.button("Назад до входу"):
            st.session_state.page = "login"
            st.rerun()
def catalog_page():
    st.title("Каталог студентських робіт")
    # Ініціалізація вкладки каталогу, якщо її немає
    if "catalog_tab" not in st.session_state:
        st.session_state.catalog_tab = "all"
    # Вкладки каталогу
    tab1, tab2, tab3 = st.tabs(["Всі роботи", "Мої роботи", "Створити роботу"])
    with tab1:
        st.session_state.catalog_tab = "all"
        st.markdown('<div class="tab-content">', unsafe_allow_html=True)
        # Пошук та фільтрація
        col1, col2, col3 = st.columns([2, 1, 1])
        with col1:
            search_query = st.text_input("Пошук за назвою або описом", key="search_all")
        with col2:
            sort_option = st.selectbox(
                "Сортувати за",
                ["Найновіші", "Найстаріші", "Найвища оцінка", "За автором"],
                key="sort_all"
            )
        with col3:
            if st.session_state.user_role == "teacher":
                filter_option = st.selectbox(
```

```
                    "Фільтрувати",
                    ["Всі", "Неоцінені", "Оцінені мною"],
                    key="filter_all"
                )
            else:
                filter_option = "Всі"
        # Отримання проектів
        projects = api_request("projects", token=st.session_state.token)
        if projects:
            # Фільтрація за пошуковим запитом
            if search_query:
                projects = [p for p in projects if search_query.lower() in p["title"].lower()
                            or (p.get("description") and search_query.lower() in
p["description"].lower())]
            # Фільтрація для викладачів
            if st.session_state.user_role == "teacher" and filter_option != "Всі":
                # Потрібно отримати оцінки поточного викладача
                # На даний момент це симуляція, оскільки у нас немає API для цього
                if filter_option == "Неоцінені":
                    projects = [p for p in projects if not p.get("avg_grade")]
                elif filter_option == "Оцінені мною":
                    # Симуляція - насправді це потрібно робити на сервері
                    projects = [p for p in projects if p.get("avg_grade")]
            # Сортування
            if sort_option == "Найновіші":
                projects = sorted(projects, key=lambda p: p.get("created_at", ""),
reverse=True)
            elif sort_option == "Найстаріші":
                projects = sorted(projects, key=lambda p: p.get("created_at", ""))
            elif sort_option == "Найвища оцінка":
                projects = sorted(projects, key=lambda p: float(p.get("avg_grade", 0) or 0),
reverse=True)
            elif sort_option == "За автором":
                projects = sorted(projects, key=lambda p: p.get("author", "").lower())
            # Статистика
            st.write(f"Знайдено робіт: {len(projects)}")
            # Відображення проектів
            if not projects:
                st.write("Проектів не знайдено.")
            else:
                for i, project in enumerate(projects):
                    display_project(project, st.session_state.user_role,
st.session_state.token, index=i)

        st.markdown('</div>', unsafe_allow_html=True)
    with tab2:
        st.session_state.catalog_tab = "my"
        st.markdown('<div class="tab-content">', unsafe_allow_html=True)
        # Фільтрація для моїх робіт
        search_query = st.text_input("Пошук серед моїх робіт", key="search_my")
        # Отримання всіх проектів
        all_projects = api_request("projects", token=st.session_state.token)
        if all_projects:
            # Фільтрація лише моїх проектів
            my_projects = [p for p in all_projects if p.get("user_id") ==
st.session_state.user_id]
            # Фільтрація за пошуковим запитом
            if search_query:
                my_projects = [p for p in my_projects if search_query.lower() in
p["title"].lower()]
            # Статистика
            st.write(f"Кількість моїх робіт: {len(my_projects)}")
            # Відображення проектів
            if not my_projects:
```

```
                st.write("У вас немає робіт. Створіть новий проект на вкладці 'Створити
роботу'.")
            else:
                for i, project in enumerate(my_projects):
                    display_project(project, st.session_state.user_role,
st.session_state.token, index=f"my_{i}")
        st.markdown('</div>', unsafe_allow_html=True)
    with tab3:
        if st.session_state.user_role in ["student", "admin"]:
            st.session_state.catalog_tab = "create"
            st.markdown('<div class="tab-content">', unsafe_allow_html=True)
            st.subheader("Створення нового проекту")
            title = st.text_input("Назва проекту", key="new_title")
            # Додамо теги для проекту
            col1, col2 = st.columns(2)
            with col1:
                course = st.selectbox("Курс", ["", "Програмування", "Математика", "Фізика",
"Хімія", "Література", "Історія"])
            with col2:
                category = st.selectbox("Категорія", ["", "Лабораторна робота", "Курсова
робота", "Дипломна робота", "Реферат", "Презентація", "Інше"])
            description = st.text_area("Опис проекту", key="new_desc", help="Детальний опис
вашого проекту. Що зроблено, які технології використані та які результати отримані.")
            uploaded_file = st.file_uploader("Завантажити файл", type=["pdf", "doc", "docx",
"zip", "rar", "txt", "ipynb", "py", "java", "cpp", "c"])
            file_path = ""
            if uploaded_file is not None:
                # Зберігаємо файл як байтовий код
                file_path = uploaded_file.name
            if st.button("Створити проект", key="btn_create"):
                if title:
                    # Перевірка назви проекту
                    if len(title) < 5:
                        st.error("Назва проекту повинна містити щонайменше 5 символів")
                        st.stop()
                    # Формування даних для проекту
                    project_description = description
                    if course or category:
                        project_description = f"Курс: {course}\nКатегорія:
{category}\n\n{description}"
                    data = {
                        "title": title,
                        "description": project_description,
                        "file_path": file_path
                    }
                    response = api_request("projects", method="POST", data=data,
token=st.session_state.token)
                    if response:
                        st.success("Проект успішно створено!")
                        time.sleep(2)
                        st.session_state.catalog_tab = "my"
                        st.rerun()
                else:
                    st.error("Назва проекту обов'язкова")
            st.markdown('</div>', unsafe_allow_html=True)
        else:
            st.warning("Тільки студенти можуть створювати нові проекти")
def project_details_page():
    project_id = st.session_state.selected_project_id
    project = api_request(f"projects/{project_id}", token=st.session_state.token)
    if not project:
        st.error("Проект не знайдено")
        if st.button("Повернутися до каталогу"):
            st.session_state.page = "catalog"
            st.rerun()
```

```
    return
# Заголовок та основна інформація
st.title(project["title"])
col1, col2 = st.columns([3, 1])
with col1:
    st.write(f"**Автор:** {project.get('author', 'Невідомо')}")
    st.write(f"**Створено:** {project.get('created_at', '')[:10]}")
with col2:
    # Кнопка "Назад до каталогу"
    if st.button("🔙 Назад до каталогу"):
        st.session_state.page = "catalog"
        st.rerun()
# Опис проекту
st.subheader("Опис проекту")
if project.get("description"):
    st.write(project["description"])
else:
    st.info("Опис проекту відсутній")
# Файл проекту
if project.get("file_path"):
    st.subheader("Файл проекту")
    st.download_button(
        label="📥 Завантажити файл",
        data=project["file_path"],
        file_name=project["file_path"].split("/")[-1],
        key=f"download_details_{project_id}"
    )
# Вкладки для оцінок, коментарів та обговорення
tab1, tab2 = st.tabs(["Оцінки та відгуки", "Коментарі та обговорення"])
with tab1:
    # Оцінки
    st.subheader("Оцінки")
    grades = project.get("grades", [])
    if not grades:
        st.info("Оцінок ще немає.")
    else:
        # Створимо таблицю оцінок
        grade_data = []
        for grade in grades:
            grade_data.append({
                "Викладач": grade.get('teacher', 'Невідомо'),
                "Оцінка": grade.get('value', 0),
                "Дата": grade.get('created_at', '')[:10]
            })
        grade_df = pd.DataFrame(grade_data)
        st.dataframe(grade_df, hide_index=True)
        # Середня оцінка
        avg_grade = sum([g.get('value', 0) for g in grades]) / len(grades)
        st.metric("Середня оцінка", f"{round(avg_grade, 1)}/100")
    # Можливість оцінити для викладачів
    if st.session_state.user_role in ["teacher", "admin"]:
        st.markdown("---")
        st.subheader("Поставити оцінку")
        grade_value = st.slider("Оцінка", 1, 100, 75)
        # Додамо поле для коментаря до оцінки
        grade_comment = st.text_area("Коментар до оцінки (опціонально)")
        if st.button("Зберегти оцінку"):
            response = api_request(f"projects/{project_id}/grade", method="POST",
                                   data={"value": grade_value}, token=st.session_state.token)
            if response:
                st.success("Оцінку збережено")
                # Якщо є коментар, додаємо його як звичайний коментар
                if grade_comment:
                    comment_text = f"**Оцінка: {grade_value}/100**\n\n{grade_comment}"
```

```
                        api_request(f"projects/{project_id}/comments", method="POST",
                                    data={"content": comment_text},
token=st.session_state.token)
                    st.rerun()
    with tab2:
        # Коментарі
        st.subheader("Коментарі")
        comments = project.get("comments", [])
        if not comments:
            st.info("Коментарів ще немає. Будьте першим, хто залишить коментар!")
        else:
            for comment in comments:
                with st.container():
                    st.markdown(f"""
                    <div class="comment">
                        <strong>{comment.get('author', 'Користувач')}</strong>
                        <small>{comment.get('created_at', '')[:16]}</small><br>
                        {comment.get('content', '')}
                    </div>
                    """, unsafe_allow_html=True)
        # Додавання коментаря
        st.markdown("---")
        st.subheader("Додати коментар")
        comment_text = st.text_area("Ваш коментар")
        if st.button("Надіслати коментар"):
            if comment_text:
                response = api_request(f"projects/{project_id}/comments", method="POST",
                                    data={"content": comment_text},
token=st.session_state.token)
                if response:
                    st.success("Коментар додано")
                    st.rerun()
            else:
                st.error("Напишіть текст коментаря")
def edit_project_page():
    project_id = st.session_state.selected_project_id
    project_data = st.session_state.edit_project_data
    st.title("Редагування проекту")
    # Розділимо опис для витягнення курсу і категорії, якщо вони є
    description = project_data.get("description", "")
    course = ""
    category = ""
    if description.startswith("Курс:"):
        lines = description.split("\n")
        if len(lines) >= 2 and lines[0].startswith("Курс:"):
            course = lines[0].replace("Курс:", "").strip()
        if len(lines) >= 2 and lines[1].startswith("Категорія:"):
            category = lines[1].replace("Категорія:", "").strip()
        # Видаляємо ці рядки з опису
        if len(lines) > 2:
            description = "\n".join(lines[2:]).strip()
    title = st.text_input("Назва проекту", value=project_data.get("title", ""))
    # Додамо теги для проекту
    col1, col2 = st.columns(2)
    with col1:
        course = st.selectbox("Курс", ["", "Програмування", "Математика", "Фізика", "Хімія",
"Література", "Історія"],
                              index=["", "Програмування", "Математика", "Фізика", "Хімія",
"Література", "Історія"].index(course) if course in ["", "Програмування", "Математика",
"Фізика", "Хімія", "Література", "Історія"] else 0)
    with col2:
        category = st.selectbox("Категорія", ["", "Лабораторна робота", "Курсова робота",
"Дипломна робота", "Реферат", "Презентація", "Інше"],
```

```
                            index=["", "Лабораторна робота", "Курсова робота", "Дипломна
робота", "Реферат", "Презентація", "Інше"].index(category) if category in ["", "Лабораторна
робота", "Курсова робота", "Дипломна робота", "Реферат", "Презентація", "Інше"] else 0)
    description_text = st.text_area("Опис проекту", value=description)
    st.write("Поточний файл:", project_data.get("file_path", "Немає файлу"))
    uploaded_file = st.file_uploader("Завантажити новий файл", type=["pdf", "doc", "docx",
"zip", "rar", "txt", "ipynb", "py", "java", "cpp", "c"])
    file_path = project_data.get("file_path", "")
    if uploaded_file is not None:
        file_path = uploaded_file.name
    col1, col2 = st.columns(2)
    with col1:
        if st.button("Зберегти зміни"):
            if title:
                # Формування опису проекту
                project_description = description_text
                if course or category:
                    project_description = f"Курс: {course}\nКатегорія:
{category}\n\n{description_text}"

                data = {
                    "title": title,
                    "description": project_description
                }
                if file_path != project_data.get("file_path", ""):
                    data["file_path"] = file_path
                response = api_request(f"projects/{project_id}", method="PUT",
                                       data=data, token=st.session_state.token)
                if response:
                    st.success("Проект успішно оновлено")
                    time.sleep(1)
                    st.session_state.page = "catalog"
                    st.rerun()
            else:
                st.error("Назва проекту обов'язкова")
    with col2:
        if st.button("Скасувати"):
            st.session_state.page = "catalog"
            st.rerun()
def chat_page():
    st.title("Чат")
    # Перевірка сесії
    if not st.session_state.get("token"):
        st.error("Сесія завершена. Будь ласка, увійдіть знову.")
        if st.button("Перейти на сторінку входу"):
            st.session_state.page = "login"
            st.rerun()
        return
    # Запит до нового API маршруту для отримання користувачів для чату
    try:
        users = api_request("chat/users", token=st.session_state.token)
        if not users:
            st.error("Сервер повернув порожній список користувачів")
            # Створюємо тимчасове рішення для демонстрації
            users = [{
                "id": 999,
                "username": "Демо-користувач",
                "role": "student"
            }]
    except Exception as e:
        st.error(f"Помилка при отриманні списку користувачів: {str(e)}")
        st.info("Перевірте, чи додано маршрут '/api/chat/users' на сервері")
        return
    # Видалення поточного користувача зі списку
    users = [u for u in users if u["id"] != st.session_state.user_id]
```

```
    if not users:
        st.write("Немає інших користувачів для спілкування.")
        return
    # Додамо фільтр для списку користувачів
    filter_options = ["Всі", "Студенти", "Викладачі", "Адміністратори"]
    filter_role = st.selectbox("Фільтрувати за роллю", filter_options)
    if filter_role != "Всі":
        role_map = {
            "Студенти": "student",
            "Викладачі": "teacher",
            "Адміністратори": "admin"
        }
        users = [u for u in users if u.get("role") == role_map[filter_role]]
    # Пошук користувачів
    search_user = st.text_input("Пошук користувача за ім'ям")
    if search_user:
        users = [u for u in users if search_user.lower() in u.get("username", "").lower()]
    # Вибір співрозмовника
    receiver_options = {f"{u.get('username', 'Користувач')} ({u.get('role', 'користувач')})":
u["id"] for u in users}
    if not receiver_options:
        st.warning("Немає користувачів, що відповідають вашому фільтру.")
        return
    col1, col2 = st.columns([3, 1])
    with col1:
        receiver_label = st.selectbox("Виберіть співрозмовника",
list(receiver_options.keys()))
    with col2:
        st.write("")
        st.write("")
        if st.button("Оновити чат"):
            st.rerun()
    receiver_id = receiver_options[receiver_label]
    # Збереження вибраного співрозмовника
    if "chat_receiver_id" not in st.session_state:
        st.session_state.chat_receiver_id = receiver_id
    elif st.session_state.chat_receiver_id != receiver_id:
        st.session_state.chat_receiver_id = receiver_id
        st.rerun()
    # Отримання повідомлень
    messages = api_request(f"messages?user_id={receiver_id}", token=st.session_state.token)
    # Відображення повідомлень
    st.subheader(f"Розмова з {receiver_label.split(' (')[0]}")
    messages_container = st.container(height=400)
    with messages_container:
        if not messages:
            st.info("Немає повідомлень. Почніть розмову зараз!")
        else:
            for message in messages:
                is_sent = message["sender_id"] == st.session_state.user_id
                message_class = "sent" if is_sent else "received"
                sender_name = "Ви" if is_sent else message.get("sender_name", "Користувач")
                st.markdown(f"""
                <div class="chat-message {message_class}">
                    <strong>{sender_name}</strong>
                    <small>{message.get('created_at', '')[:16]}</small><br>
                    {message.get('content', '')}
                </div>
                """, unsafe_allow_html=True)
    # Поле для нового повідомлення
    st.subheader("Нове повідомлення")
    # Використовуємо інший ключ для збереження тексту повідомлення
    if "message_text" not in st.session_state:
        st.session_state.message_text = ""
```

```
    message_text = st.text_area("Введіть повідомлення", key="new_message",
value=st.session_state.message_text)
    col1, col2 = st.columns([3, 1])
    with col1:
        if st.button("Надіслати", use_container_width=True):
            if message_text:
                response = api_request("messages", method="POST", data={
                    "receiver_id": receiver_id,
                    "content": message_text
                }, token=st.session_state.token)
                if response:
                    st.success("Повідомлення надіслано")
                    # Оновлюємо текст для наступного вводу перед створенням віджета
                    st.session_state.message_text = ""
                    st.rerun()
            else:
                st.error("Введіть текст повідомлення")
def profile_page():
    st.title("Особистий кабінет")
    # Отримання інформації про профіль
    profile = api_request("profile", token=st.session_state.token)
    if not profile:
        st.error("Не вдалося отримати дані профілю")
        return
    # Вкладки для різних секцій профілю
    tab1, tab2 = st.tabs(["Інформація", "Редагування профілю"])
    with tab1:
        # Базова інформація про користувача
        st.markdown(f"""
        <div class="profile-card">
            <h3>{profile.get('username', '')}</h3>
            <p><strong>Email:</strong> {profile.get('email', '')}</p>
            <p><strong>Роль:</strong> {profile.get('role', '')}</p>
            <p><strong>Дата реєстрації:</strong> {profile.get('created_at', '')[:10]}</p>
        </div>
        """, unsafe_allow_html=True)
        # Отримуємо проекти користувача
        all_projects = api_request("projects", token=st.session_state.token)
        if all_projects:
            my_projects = [p for p in all_projects if p.get("user_id") ==
st.session_state.user_id]
            if profile.get('role') == 'student':
                st.subheader("Мої проекти")
                st.write(f"Кількість проектів: {len(my_projects)}")
                # Останні проекти
                if my_projects:
                    st.subheader("Останні проекти")
                    recent_projects = sorted(my_projects, key=lambda p: p.get("created_at",
""), reverse=True)[:2]
                    for i, project in enumerate(recent_projects):
                        display_project(project, st.session_state.user_role,
st.session_state.token, show_details=False, index=f"profile_{i}")
    with tab2:
        st.subheader("Редагування профілю")
        email = st.text_input("Новий Email", value=profile.get("email", ""))
        st.subheader("Зміна пароля")
        current_password = st.text_input("Поточний пароль", type="password")
        new_password = st.text_input("Новий пароль", type="password")
        confirm_password = st.text_input("Підтвердження нового пароля", type="password")
        if st.button("Зберегти зміни"):
            data = {}
            if email != profile.get("email", ""):
                data["email"] = email
            if current_password and new_password and confirm_password:
                if new_password != confirm_password:
```

```
                    st.error("Паролі не співпадають")
                    st.stop()
                if len(new_password) < 6:
                    st.error("Новий пароль має містити щонайменше 6 символів")
                    st.stop()
                data["current_password"] = current_password
                data["new_password"] = new_password
            if data:
                response = api_request("profile", method="PUT", data=data,
token=st.session_state.token)
                if response:
                    st.success("Профіль успішно оновлено")
                    st.rerun()
            else:
                st.warning("Немає змін для збереження")
def admin_page():
    if st.session_state.user_role != "admin":
        st.error("Доступ заборонено")
        return
    st.title("Адміністрування")
    # Вкладки для різних функцій адміністрування
    tab1, tab2 = st.tabs(["Користувачі", "Проекти"])
    with tab1:
        st.subheader("Управління користувачами")
        # Отримання списку користувачів
        users = api_request("users", token=st.session_state.token)
        if not users:
            st.error("Не вдалося отримати список користувачів")
            return
        # Фільтри для користувачів
        col1, col2 = st.columns(2)
        with col1:
            role_filter = st.selectbox("Фільтр за роллю", ["Всі", "student", "teacher",
"admin"])
        with col2:
            search_user = st.text_input("Пошук за ім'ям або email")
        # Застосовуємо фільтри
        filtered_users = users
        if role_filter != "Всі":
            filtered_users = [u for u in filtered_users if u["role"] == role_filter]
        if search_user:
            filtered_users = [u for u in filtered_users if search_user.lower() in
u["username"].lower() or
                              search_user.lower() in u.get("email", "").lower()]
        # Відображення користувачів у таблиці
        user_data = []
        for user in filtered_users:
            user_data.append({
                "ID": user["id"],
                "Логін": user["username"],
                "Email": user["email"],
                "Роль": user["role"],
                "Створено": user["created_at"][:10]
            })
        if user_data:
            df = pd.DataFrame(user_data)
            st.dataframe(df, hide_index=True, use_container_width=True)
            # Видалення користувача
            st.subheader("Видалення користувача")
            user_to_delete = st.selectbox(
                "Виберіть користувача для видалення",
                [f"{u['username']} (ID: {u['id']})" for u in filtered_users if u["id"] !=
st.session_state.user_id]
            )
            if st.button("Видалити користувача"):
```

```
                user_id = int(user_to_delete.split("ID: ")[1].strip(")"))
                if st.session_state.user_id == user_id:
                    st.error("Ви не можете видалити свій обліковий запис")
                    st.stop()
                confirm = st.checkbox("Підтвердити видалення")
                if confirm:
                    response = api_request(f"users/{user_id}", method="DELETE",
token=st.session_state.token)
                    if response:
                        st.success(f"Користувача видалено")
                        st.rerun()
        else:
            st.write("Немає користувачів, що відповідають вашим критеріям.")
    with tab2:
        st.subheader("Управління проектами")
        # Отримання всіх проектів
        projects = api_request("projects", token=st.session_state.token)
        if not projects:
            st.error("Не вдалося отримати проекти")
            return
        # Фільтри для проектів
        col1, col2 = st.columns(2)
        with col1:
            search_project = st.text_input("Пошук за назвою проекту")
        with col2:
            sort_option = st.selectbox(
                "Сортувати за",
                ["Найновіші", "Найстаріші", "Найвища оцінка", "За автором"]
            )
        # Застосовуємо фільтри
        if search_project:
            projects = [p for p in projects if search_project.lower() in p["title"].lower()]
        # Сортування
        if sort_option == "Найновіші":
            projects = sorted(projects, key=lambda p: p.get("created_at", ""), reverse=True)
        elif sort_option == "Найстаріші":
            projects = sorted(projects, key=lambda p: p.get("created_at", ""))
        elif sort_option == "Найвища оцінка":
            projects = sorted(projects, key=lambda p: float(p.get("avg_grade", 0) or 0),
reverse=True)
        elif sort_option == "За автором":
            projects = sorted(projects, key=lambda p: p.get("author", "").lower())
        # Відображення проектів у таблиці
        project_data = []
        for project in projects:
            project_data.append({
                "ID": project["id"],
                "Назва": project["title"],
                "Автор": project.get("author", "Невідомо"),
                "Середня оцінка": round(float(project.get("avg_grade", 0) or 0), 1),
                "Коментарі": project.get("comments_count", 0),
                "Створено": project.get("created_at", "")[:10]
            })
        if project_data:
            df = pd.DataFrame(project_data)
            st.dataframe(df, hide_index=True, use_container_width=True)
            # Видалення проекту
            st.subheader("Видалення проекту")
            project_to_delete = st.selectbox(
                "Виберіть проект для видалення",
                [f"{p['title']} (ID: {p['id']})" for p in projects]
            )
            if st.button("Видалити проект"):
                project_id = int(project_to_delete.split("ID: ")[1].strip(")"))
                confirm = st.checkbox("Підтвердити видалення проекту")
```

```
                if confirm:
                    response = api_request(f"projects/{project_id}", method="DELETE",
token=st.session_state.token)
                    if response:
                        st.success(f"Проект видалено")
                        st.rerun()
        else:
            st.write("Немає проектів, що відповідають вашим критеріям.")
# Головна функція програми
def main():
    # Застосування CSS стилів
    apply_css()
    # Ініціалізація стану сесії
    if "page" not in st.session_state:
        st.session_state.page = "login"
    # Бічна панель для авторизованих користувачів
    if st.session_state.get("token"):
        with st.sidebar:
            st.write(f"**Вітаємо, {st.session_state.username}!**")
            st.write(f"Роль: {st.session_state.user_role}")
            st.markdown("---")
            if st.button("Каталог", use_container_width=True):
                st.session_state.page = "catalog"
                st.rerun()
            if st.button("Чат", use_container_width=True):
                st.session_state.page = "chat"
                st.rerun()
            if st.button("Особистий кабінет", use_container_width=True):
                st.session_state.page = "profile"
                st.rerun()
            if st.session_state.user_role == "admin":
                if st.button("Адміністрування", use_container_width=True):
                    st.session_state.page = "admin"
                    st.rerun()
            st.markdown("---")
            if st.button("Вийти", use_container_width=True):
                for key in list(st.session_state.keys()):
                    del st.session_state[key]
                st.session_state.page = "login"
                st.rerun()
            # Додаткова інформація внизу сайдбару
            st.markdown("---")
            st.markdown("""
            <small>Система каталогізації студентських робіт</small><br>
            <small>Версія 1.0</small>
            """, unsafe_allow_html=True)
    # Вибір сторінки для відображення
    if st.session_state.page == "login":
        login_page()
    elif st.session_state.page == "register":
        register_page()
    elif st.session_state.page == "catalog":
        catalog_page()
    elif st.session_state.page == "project_details":
        project_details_page()
    elif st.session_state.page == "create_project":
        create_project_page()
    elif st.session_state.page == "edit_project":
        edit_project_page()
    elif st.session_state.page == "chat":
        chat_page()
    elif st.session_state.page == "profile":
        profile_page()
    elif st.session_state.page == "admin":
        admin_page()
```

```
if __name__ == "__main__":
    main()
```

seed.py

```
import sqlite3
import hashlib
import datetime
import os
import random
# Путь к базе данных
DATABASE = 'database.db'
# Функция для хеширования пароля
def hash_password(password):
    return hashlib.sha256(password.encode()).hexdigest()
# Создаем новую базу данных (удаляем старую, если существует)
if os.path.exists(DATABASE):
    os.remove(DATABASE)
# Подключаемся к базе данных
conn = sqlite3.connect(DATABASE)
cursor = conn.cursor()
# Создаем таблицы (скопировано из server.py)
cursor.execute('''
CREATE TABLE users (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    username TEXT UNIQUE NOT NULL,
    email TEXT UNIQUE NOT NULL,
    password TEXT NOT NULL,
    role TEXT NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
)
''')
cursor.execute('''
CREATE TABLE projects (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    title TEXT NOT NULL,
    description TEXT,
    file_path TEXT,
    user_id INTEGER NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES users (id)
)
''')
cursor.execute('''
CREATE TABLE comments (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    content TEXT NOT NULL,
    user_id INTEGER NOT NULL,
    project_id INTEGER NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES users (id),
    FOREIGN KEY (project_id) REFERENCES projects (id)
)
''')
cursor.execute('''
CREATE TABLE grades (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    value INTEGER NOT NULL,
    user_id INTEGER NOT NULL,
    project_id INTEGER NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES users (id),
```

```
    FOREIGN KEY (project_id) REFERENCES projects (id)
)
''')
cursor.execute('''
CREATE TABLE messages (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    content TEXT NOT NULL,
    sender_id INTEGER NOT NULL,
    receiver_id INTEGER NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (sender_id) REFERENCES users (id),
    FOREIGN KEY (receiver_id) REFERENCES users (id)
)
''')
print("Таблиці створено.")
# Добавляем админа
admin = {
    "username": "admin",
    "email": "admin@university.edu",
    "password": hash_password("admin123"),
    "role": "admin"
}
cursor.execute(
    'INSERT INTO users (username, email, password, role) VALUES (?, ?, ?, ?)',
    (admin["username"], admin["email"], admin["password"], admin["role"])
)
print("Адміністратора додано.")
# Добавляем преподавателей
teachers = [
    {
        "username": "petrov_ivan",
        "email": "petrov.ivan@university.edu",
        "password": hash_password("teacher1"),
        "role": "teacher",
        "full_name": "Іван Петров"
    },
    {
        "username": "koval_maria",
        "email": "koval.maria@university.edu",
        "password": hash_password("teacher2"),
        "role": "teacher",
        "full_name": "Марія Коваль"
    },
    {
        "username": "shevchenko_olga",
        "email": "shevchenko.olga@university.edu",
        "password": hash_password("teacher3"),
        "role": "teacher",
        "full_name": "Ольга Шевченко"
    }
]
for teacher in teachers:
    cursor.execute(
        'INSERT INTO users (username, email, password, role) VALUES (?, ?, ?, ?)',
        (teacher["username"], teacher["email"], teacher["password"], teacher["role"])
    )
print("Викладачів додано.")
# Добавляем студентов
students = [
    {"username": "bondarenko_andriy", "email": "bondarenko.andriy@university.edu",
"full_name": "Андрій Бондаренко"},
    {"username": "tymoshenko_olena", "email": "tymoshenko.olena@university.edu", "full_name":
"Олена Тимошенко"},
    {"username": "ivanenko_dmytro", "email": "ivanenko.dmytro@university.edu", "full_name":
"Дмитро Іваненко"},
```

```
    {"username": "melnyk_sophie", "email": "melnyk.sophie@university.edu", "full_name":
"Софія Мельник"},
    {"username": "kravchuk_roman", "email": "kravchuk.roman@university.edu", "full_name":
"Роман Кравчук"},
    {"username": "ponomarenko_anna", "email": "ponomarenko.anna@university.edu", "full_name":
"Анна Пономаренко"},
    {"username": "vasyliuk_oleg", "email": "vasyliuk.oleg@university.edu", "full_name": "Олег
Василюк"},
    {"username": "lysenko_daria", "email": "lysenko.daria@university.edu", "full_name":
"Дарія Лисенко"},
    {"username": "stepanenko_yuriy", "email": "stepanenko.yuriy@university.edu", "full_name":
"Юрій Степаненко"},
    {"username": "kovalchuk_iryna", "email": "kovalchuk.iryna@university.edu", "full_name":
"Ірина Ковальчук"},
    {"username": "hrytsenko_maksym", "email": "hrytsenko.maksym@university.edu", "full_name":
"Максим Гриценко"},
    {"username": "levchenko_natalia", "email": "levchenko.natalia@university.edu",
"full_name": "Наталія Левченко"},
    {"username": "ponomarenko_oleksandr", "email": "ponomarenko.oleksandr@university.edu",
"full_name": "Олександр Пономаренко"},
    {"username": "sydorenko_tetiana", "email": "sydorenko.tetiana@university.edu",
"full_name": "Тетяна Сидоренко"},
    {"username": "fedorenko_victor", "email": "fedorenko.victor@university.edu", "full_name":
"Віктор Федоренко"},
    {"username": "kravets_yulia", "email": "kravets.yulia@university.edu", "full_name": "Юлія
Кравець"},
    {"username": "soroka_pavlo", "email": "soroka.pavlo@university.edu", "full_name": "Павло
Сорока"},
    {"username": "marchenko_diana", "email": "marchenko.diana@university.edu", "full_name":
"Діана Марченко"},
    {"username": "kozak_taras", "email": "kozak.taras@university.edu", "full_name": "Тарас
Козак"},
    {"username": "nesterenko_kateryna", "email": "nesterenko.kateryna@university.edu",
"full_name": "Катерина Нестеренко"}
]
for student in students:
    cursor.execute(
        'INSERT INTO users (username, email, password, role) VALUES (?, ?, ?, ?)',
        (student["username"], student["email"], hash_password("student123"), "student")
    )
print("Студентів додано.")
# Курсы и категории
courses = ["Програмування", "Математика", "Фізика", "Хімія", "Література", "Історія"]
categories = ["Лабораторна робота", "Курсова робота", "Дипломна робота", "Реферат",
"Презентація"]
# Проекты программирования
programming_projects = [
    {
        "title": "Розробка веб-додатка для управління студентськими проектами",
        "description": """Курс: Програмування
Категорія: Курсова робота
Веб-додаток для каталогізації та управління студентськими проектами з можливістю завантаження
файлів, коментування та оцінювання. Розроблений з використанням Flask, SQLite і Streamlit.
Реалізовано систему аутентифікації з різними рівнями доступу, інтерфейс перегляду та пошуку
проектів, а також функції для коментування та оцінювання.""",
        "file_path": "project_management_app.zip"
    },
    {
        "title": "Алгоритми сортування та їх порівняльний аналіз",
        "description": """Курс: Програмування
Категорія: Лабораторна робота
Реалізація та порівняльний аналіз різних алгоритмів сортування (бульбашкове сортування,
швидке сортування, сортування злиттям) на різних типах даних. Включає графіки продуктивності
та аналіз складності алгоритмів для різних розмірів вхідних даних.""",
        "file_path": "sorting_algorithms.py"
```

```
    },
    {
        "title": "Розробка мобільного додатка для розкладу занять",
        "description": """Курс: Програмування
Категорія: Дипломна робота
Мобільний додаток для перегляду та управління розкладом занять. Включає сповіщення про
початок пар, інтеграцію з календарем та можливість синхронізації між різними пристроями.
Розроблений з використанням Flutter і Firebase.""",
        "file_path": "schedule_app.zip"
    },
    {
        "title": "Система розпізнавання облич для контролю відвідуваності",
        "description": """Курс: Програмування
Категорія: Курсова робота
Система, що використовує комп'ютерний зір для розпізнавання облич студентів та автоматичного
відзначення їхньої присутності на заняттях. Розроблено з використанням Python, OpenCV та
бібліотеки dlib.""",
        "file_path": "face_recognition.py"
    },
    {
        "title": "Розробка чат-бота для навчання програмуванню",
        "description": """Курс: Програмування
Категорія: Лабораторна робота
Телеграм-бот, що допомагає вивчати програмування через інтерактивні уроки та практичні
завдання з автоматичною перевіркою. Підтримує мови Python, JavaScript та C++.""",
        "file_path": "programming_bot.py"
    },
    {
        "title": "Реалізація нейронної мережі для класифікації зображень",
        "description": """Курс: Програмування
Категорія: Курсова робота
Розробка нейронної мережі для класифікації зображень з використанням TensorFlow. Модель
навчена на наборі даних CIFAR-10 та досягає точності 87%.""",
        "file_path": "neural_network.py"
    },
    {
        "title": "Створення API для університетської бібліотеки",
        "description": """Курс: Програмування
Категорія: Лабораторна робота
REST API для університетської бібліотеки, що дозволяє шукати книги, керувати запозиченнями та
резервувати матеріали. Розроблений з використанням Django REST Framework.""",
        "file_path": "library_api.py"
    },
    {
        "title": "Розробка гри 'Змійка' на Python",
        "description": """Курс: Програмування
Категорія: Лабораторна робота
Класична гра 'Змійка' з використанням бібліотеки Pygame. Включає рекорди, різні рівні
складності та кілька режимів гри.""",
        "file_path": "snake_game.py"
    }
]
# Проекты математики
math_projects = [
    {
        "title": "Дослідження методів чисельного інтегрування",
        "description": """Курс: Математика
Категорія: Курсова робота
Порівняльний аналіз різних методів чисельного інтегрування: метод прямокутників, метод
трапецій, метод Сімпсона. Включає оцінку похибок та оптимізацію алгоритмів.""",
        "file_path": "numerical_integration.pdf"
    },
    {
        "title": "Застосування теорії графів для оптимізації маршрутів",
        "description": """Курс: Математика
```

```
Категорія: Лабораторна робота
Використання алгоритмів теорії графів (Дейкстра, Краскал, Форд-Фалкерсон) для знаходження
оптимальних маршрутів між корпусами університету.""",
        "file_path": "graph_theory.pdf"
    },
    {
        "title": "Розв'язання диференціальних рівнянь методом Рунге-Кутта",
        "description": """Курс: Математика
Категорія: Лабораторна робота
Реалізація та аналіз методу Рунге-Кутта для чисельного розв'язання звичайних диференціальних
рівнянь. Порівняння з аналітичними рішеннями.""",
        "file_path": "runge_kutta.pdf"
    },
    {
        "title": "Дослідження фрактальних структур",
        "description": """Курс: Математика
Категорія: Курсова робота
Дослідження математичних властивостей фракталів, включаючи множину Мандельброта та множину
Жюліа. Візуалізація фракталів з використанням Python.""",
        "file_path": "fractals.pdf"
    },
    {
        "title": "Статистичний аналіз успішності студентів",
        "description": """Курс: Математика
Категорія: Дипломна робота

Аналіз факторів, що впливають на успішність студентів, з використанням методів статистичного
аналізу та машинного навчання.""",
        "file_path": "statistical_analysis.pdf"
    },
    {
        "title": "Модель прогнозування цін на ринку нерухомості",
        "description": """Курс: Математика
Категорія: Курсова робота
Розробка математичної моделі для прогнозування цін на нерухомість з використанням
регресійного аналізу та нейронних мереж.""",
        "file_path": "price_prediction.pdf"
    }
]
# Проекты физики
physics_projects = [
    {
        "title": "Дослідження руху тіл у гравітаційному полі",
        "description": """Курс: Фізика
Категорія: Лабораторна робота
Експериментальне дослідження руху тіл у гравітаційному полі Землі та порівняння результатів з
теоретичними розрахунками.""",
        "file_path": "gravitational_field.pdf"
    },
    {
        "title": "Моделювання коливальних процесів",
        "description": """Курс: Фізика
Категорія: Курсова робота

Розробка математичної моделі та комп'ютерне моделювання різних видів коливань: вільних,
затухаючих та вимушених.""",
        "file_path": "oscillations.pdf"
    },
    {
        "title": "Дослідження властивостей напівпровідників",
        "description": """Курс: Фізика
Категорія: Лабораторна робота
Експериментальне дослідження електричних властивостей напівпровідників та їх залежності від
температури.""",
        "file_path": "semiconductors.pdf"
```

```
    },
    {
        "title": "Взаємодія електромагнітних хвиль з речовиною",
        "description": """Курс: Фізика
Категорія: Курсова робота
Теоретичне та експериментальне дослідження взаємодії електромагнітних хвиль різних частот з
різними типами матеріалів.""",
        "file_path": "electromagnetic_waves.pdf"
    },
    {
        "title": "Розрахунок параметрів оптичної системи телескопа",
        "description": """Курс: Фізика
Категорія: Лабораторна робота
Проектування та розрахунок параметрів оптичної системи телескопа-рефлектора з заданими
характеристиками.""",
        "file_path": "telescope_design.pdf"
    }
]
# Проекты химии
chemistry_projects = [
    {
        "title": "Дослідження кінетики хімічних реакцій",
        "description": """Курс: Хімія
Категорія: Лабораторна робота
Експериментальне дослідження швидкості хімічних реакцій та факторів, що на неї впливають:
концентрації, температури, каталізаторів.""",
        "file_path": "chemical_kinetics.pdf"
    },
    {
        "title": "Синтез та дослідження властивостей нанокомпозитів",
        "description": """Курс: Хімія
Категорія: Дипломна робота
Розробка методики синтезу нанокомпозитних матеріалів та дослідження їх структурних,
механічних та електрофізичних властивостей.""",
        "file_path": "nanocomposites.pdf"
    },
    {
        "title": "Аналіз якості питної води",
        "description": """Курс: Хімія
Категорія: Курсова робота
Хімічний аналіз зразків питної води з різних джерел та порівняння результатів з санітарними
нормами.""",
        "file_path": "water_analysis.pdf"
    },
    {
        "title": "Електрохімічні методи аналізу",
        "description": """Курс: Хімія
Категорія: Лабораторна робота
Застосування електрохімічних методів (потенціометрія, вольтамперометрія) для аналізу різних
речовин.""",
        "file_path": "electrochemical_analysis.pdf"
    }
]
# Проекты литературы
literature_projects = [
    {
        "title": "Образ головного героя в романі І. Франка 'Перехресні стежки'",
        "description": """Курс: Література
Категорія: Курсова робота
Аналіз образу головного героя роману Івана Франка 'Перехресні стежки' в контексті соціально-
історичних умов того часу.""",
        "file_path": "franko_analysis.pdf"
    },
    {
        "title": "Символізм у поезії Лесі Українки",
```

```
        "description": """Курс: Література
Категорія: Реферат
Дослідження символіки та образної системи в поетичних творах Лесі Українки.""",
        "file_path": "ukrainka_symbolism.pdf"
    },
    {
        "title": "Порівняльний аналіз творів Т. Шевченка та М. Гоголя",
        "description": """Курс: Література
Категорія: Курсова робота
Порівняльний аналіз творчості Тараса Шевченка та Миколи Гоголя в контексті українського
національного відродження.""",
        "file_path": "shevchenko_gogol.pdf"
    },
    {
        "title": "Постмодернізм в українській літературі",
        "description": """Курс: Література
Категорія: Презентація
Огляд постмодерністських тенденцій в сучасній українській літературі на прикладі творів Ю.
Андруховича, О. Забужко, С. Жадана.""",
        "file_path": "postmodernism.pptx"
    }
]

# Проекты истории
history_projects = [
    {
        "title": "Київська Русь: формування та розвиток держави",
        "description": """Курс: Історія
Категорія: Курсова робота
Дослідження процесу формування та розвитку Київської Русі як середньовічної
східнослов'янської держави.""",
        "file_path": "kyivan_rus.pdf"
    },
    {
        "title": "Українське козацтво в XVI-XVII століттях",
        "description": """Курс: Історія
Категорія: Реферат
Аналіз ролі та значення українського козацтва в історії України XVI-XVII століть.""",
        "file_path": "cossacks.pdf"
    },
    {
        "title": "Українська революція 1917-1921 років",
        "description": """Курс: Історія
Категорія: Презентація
Презентація про основні етапи та події Української революції 1917-1921 років.""",
        "file_path": "ukrainian_revolution.pptx"
    },
    {
        "title": "Голодомор 1932-1933 років: причини та наслідки",
        "description": """Курс: Історія
Категорія: Курсова робота
Дослідження причин, перебігу та наслідків Голодомору 1932-1933 років в Україні.""",
        "file_path": "holodomor.pdf"
    }
]
# Объединение всех проектов
all_projects = programming_projects + math_projects + physics_projects + chemistry_projects +
literature_projects + history_projects
# Добавление проектов в базу данных
# Получаем всех студентов
cursor.execute('SELECT id FROM users WHERE role = "student"')
student_ids = [row[0] for row in cursor.fetchall()]
# Создаем проекты
for i, project in enumerate(all_projects):
    # Выбираем случайного студента
```

```
    student_id = random.choice(student_ids)
    # Создаем дату (чтобы проекты были распределены по времени)
    days_ago = random.randint(1, 100)
    project_date = (datetime.datetime.now() -
datetime.timedelta(days=days_ago)).strftime("%Y-%m-%d %H:%M:%S")
    cursor.execute(
        'INSERT INTO projects (title, description, file_path, user_id, created_at) VALUES
(?, ?, ?, ?, ?)',
        (project["title"], project["description"], project["file_path"], student_id,
project_date)
    )
print(f"Додано {len(all_projects)} проектів.")
# Добавление оценок
cursor.execute('SELECT id FROM users WHERE role = "teacher"')
teacher_ids = [row[0] for row in cursor.fetchall()]
cursor.execute('SELECT id FROM projects')
project_ids = [row[0] for row in cursor.fetchall()]
# Оценка примерно 70% проектов
projects_to_grade = random.sample(project_ids, int(len(project_ids) * 0.7))
for project_id in projects_to_grade:
    # Сколько учителей оценят проект (1-3)
    num_teachers = random.randint(1, min(3, len(teacher_ids)))
    grading_teachers = random.sample(teacher_ids, num_teachers)
    for teacher_id in grading_teachers:
        # Оценка от 60 до 100
        grade_value = random.randint(60, 100)
        # Дата оценки (после создания проекта)
        cursor.execute('SELECT created_at FROM projects WHERE id = ?', (project_id,))
        project_date = cursor.fetchone()[0]
        days_after = random.randint(1, 14)
        grade_date = (datetime.datetime.strptime(project_date, "%Y-%m-%d %H:%M:%S") +
                      datetime.timedelta(days=days_after)).strftime("%Y-%m-%d %H:%M:%S")
        cursor.execute(
            'INSERT INTO grades (value, user_id, project_id, created_at) VALUES (?, ?, ?, ?)',
            (grade_value, teacher_id, project_id, grade_date)
        )
print("Оцінки додано.")
# Добавление комментариев
comments_data = [
    # Положительные комментарии от преподавателей
    [
        "Відмінна робота! Особливо вражає глибина дослідження теми.",
        "Дуже добре структурована робота. Чітко видно логіку дослідження.",
        "Цікавий підхід до розв'язання проблеми. Продовжуйте в тому ж дусі.",
        "Робота демонструє глибоке розуміння предмету дослідження.",
        "Чудова презентація результатів. Всі діаграми інформативні та добре оформлені.",
        "Ваша робота виділяється серед інших оригінальністю підходу.",
        "Дуже ретельне опрацювання джерел. Видно, що Ви провели ґрунтовне дослідження.",
        "Ви чудово впоралися з поставленим завданням. Результати перевершили очікування."
    ],
    # Конструктивная критика от преподавателей
    [
        "Робота загалом хороша, але рекомендую додати більше прикладів для ілюстрації
ключових моментів.",
        "Варто розширити розділ методології. Не зовсім зрозуміло, як саме проводилося
дослідження.",
        "Деякі висновки потребують додаткового обґрунтування. Рекомендую переглянути цей
аспект.",
        "Текст містить кілька граматичних помилок, які варто виправити перед фінальною
версією.",
        "Бібліографія могла б бути більш розширеною. Рекомендую додати кілька сучасних
джерел.",
        "Структура роботи потребує деякого удосконалення, особливо в розділі аналізу
результатів.",
        "Графіки інформативні, але підписи до них занадто малі та нечіткі.",
```

```
        "Добра робота, але висновки могли б бути більш конкретними та пов'язаними з метою
дослідження."
    ],
    # Комментарии от студентов
    [
        "Дуже цікава тема дослідження! Я також працюю над подібною проблемою.",
        "Мені сподобався Ваш підхід до аналізу даних. Можна дізнатися, яке програмне
забезпечення Ви використовували?",
        "Чудова презентація матеріалу. Особливо корисними були приклади з реального життя.",
        "Дякую за детальний опис методології. Це дуже допомогло мені з власним проектом.",
        "Можете поділитися посиланнями на деякі джерела, які Ви використовували? Це було б
дуже корисно.",
        "Ваша робота надихнула мене на власне дослідження в цій сфері.",
        "Чи плануєте Ви продовжувати дослідження цієї теми? Результати дуже перспективні.",
        "Чи можна використати Вашу методологію для аналізу інших подібних проблем?"
    ]
]
# Комментирование примерно 60% проектов
projects_to_comment = random.sample(project_ids, int(len(project_ids) * 0.6))
cursor.execute('SELECT id FROM users')
all_user_ids = [row[0] for row in cursor.fetchall()]
for project_id in projects_to_comment:
    # Количество комментариев для этого проекта (1-5)
    num_comments = random.randint(1, 5)
    # Дата создания проекта
    cursor.execute('SELECT created_at FROM projects WHERE id = ?', (project_id,))
    project_date = cursor.fetchone()[0]
    project_date_obj = datetime.datetime.strptime(project_date, "%Y-%m-%d %H:%M:%S")
    for i in range(num_comments):
        # Выбираем случайного пользователя
        user_id = random.choice(all_user_ids)
        # Определяем тип комментария в зависимости от роли пользователя
        cursor.execute('SELECT role FROM users WHERE id = ?', (user_id,))
        user_role = cursor.fetchone()[0]
        if user_role == "teacher":
            if random.random() < 0.7:  # 70% положительных, 30% с критикой
                comment_content = random.choice(comments_data[0])
            else:
                comment_content = random.choice(comments_data[1])
        else:  # student or admin
            comment_content = random.choice(comments_data[2])
        # Дата комментария (после создания проекта)
        days_after = random.randint(1, 30)
        comment_date = (project_date_obj + datetime.timedelta(days=days_after)).strftime("%Y-
%m-%d %H:%M:%S")
        cursor.execute(
            'INSERT INTO comments (content, user_id, project_id, created_at) VALUES
(?, ?, ?, ?)',
            (comment_content, user_id, project_id, comment_date)
        )
print("Коментарі додано.")
# Добавление сообщений чата
chat_messages = [
    # Сообщения от студентов к преподавателям
    [
        "Добрий день! Чи можна дізнатися про терміни здачі курсового проекту?",
        "Вітаю! Хотів би проконсультуватися щодо вибору теми для курсової роботи.",
        "Доброго дня! Чи можу я відправити Вам чернетку мого проекту для попереднього
перегляду?",
        "Вибачте за турботу, але я хотів уточнити вимоги до оформлення роботи.",
        "Добрий день! Чи можна перенести дату захисту мого проекту? Виникли непередбачені
обставини.",
        "Підкажіть, будь ласка, які джерела краще використовувати для мого дослідження?",
        "Дякую за вчорашню консультацію! Вона була дуже корисною для мого проекту."
    ],
```

```
    # Ответы преподавателей студентам
    [
        "Добрий день! Термін здачі курсових проектів - 15 травня. Детальну інформацію я
розіслав на електронні пошти.",
        "Вітаю! Звичайно, давайте обговоримо вибір теми. Які напрямки вас цікавлять?",
        "Доброго дня! Так, звісно, надсилайте. Я перегляну та дам рекомендації.",
        "Вимоги до оформлення доступні на сайті кафедри у розділі 'Студентам'. Також я можу
надіслати вам шаблон.",
        "Добрий день! Так, це можливо. Будь ласка, напишіть офіційну заяву та вкажіть
причину.",
        "Рекомендую почати з цих джерел: [список наукових праць]. Також перевірте публікації
в журналі нашого університету.",
        "Завжди радий допомогти! Якщо виникнуть додаткові питання, не соромтеся звертатися."
    ],
    # Сообщения между студентами
    [
        "Привіт! Ти вже обрав тему для курсової?",
        "Вітаю! Чи не міг би ти пояснити, як працює ця формула в завданні?",
        "Привіт! Коли у нас дедлайн з проекту по програмуванню?",
        "Слухай, ти не знаєш, як оформлювати бібліографію за новими вимогами?",
        "Привіт! Як готуєшся до захисту проекту?",
        "Вітаю! Можеш поділитися своїми конспектами з останньої лекції? Я, на жаль,
пропустив.",
        "Слухай, давай разом підготуємося до захисту проектів? Можемо попрактикуватися у
презентації."
    ]
]
# Создаем некоторые сообщения чата
for i in range(30):  # Создаем 30 диалогов
    # Выбираем отправителя и получателя
    sender_id = random.choice(all_user_ids)
    # Получатель не должен быть отправителем
    receiver_candidates = [uid for uid in all_user_ids if uid != sender_id]
    receiver_id = random.choice(receiver_candidates)
    # Определяем роли отправителя и получателя
    cursor.execute('SELECT role FROM users WHERE id = ?', (sender_id,))
    sender_role = cursor.fetchone()[0]
    cursor.execute('SELECT role FROM users WHERE id = ?', (receiver_id,))
    receiver_role = cursor.fetchone()[0]
    # Выбираем сообщение в зависимости от ролей
    if sender_role == "student" and receiver_role == "teacher":
        message_content = random.choice(chat_messages[0])
    elif sender_role == "teacher" and receiver_role == "student":
        message_content = random.choice(chat_messages[1])
    else:  # student to student or other combinations
        message_content = random.choice(chat_messages[2])
    # Дата сообщения (последние 30 дней)
    days_ago = random.randint(0, 30)
    message_date = (datetime.datetime.now() -
datetime.timedelta(days=days_ago)).strftime("%Y-%m-%d %H:%M:%S")
    cursor.execute(
        'INSERT INTO messages (content, sender_id, receiver_id, created_at) VALUES
(?, ?, ?, ?)',
        (message_content, sender_id, receiver_id, message_date)
    )
    # Создаем отдельные ответы на некоторые сообщения (50% вероятность)
    if random.random() < 0.5:
        # Меняем отправителя и получателя местами
        reply_sender_id = receiver_id
        reply_receiver_id = sender_id
        # Выбираем соответствующий ответ
        if receiver_role == "student" and sender_role == "teacher":
            reply_content = random.choice(chat_messages[0])
        elif receiver_role == "teacher" and sender_role == "student":
            reply_content = random.choice(chat_messages[1])
```

```
        else:
            reply_content = random.choice(chat_messages[2])
        # Дата ответа (после даты первого сообщения)
        message_date_obj = datetime.datetime.strptime(message_date, "%Y-%m-%d %H:%M:%S")
        hours_after = random.randint(1, 48)
        reply_date = (message_date_obj + datetime.timedelta(hours=hours_after)).strftime("%Y-
%m-%d %H:%M:%S")
        cursor.execute(
            'INSERT INTO messages (content, sender_id, receiver_id, created_at) VALUES
(?, ?, ?, ?)',
            (reply_content, reply_sender_id, reply_receiver_id, reply_date)
        )
print("Повідомлення чату додано.")
# Сохраняем изменения и закрываем соединение
conn.commit()
conn.close()
print("База даних успішно заповнена.")
print(f"Загалом: 1 адміністратор, {len(teachers)} викладачів, {len(students)} студентів,
{len(all_projects)} проектів")
```

server.py

```
from flask import Flask, request, jsonify, g
from flask_cors import CORS
import sqlite3
import hashlib
import secrets
import jwt
import datetime
import os
from functools import wraps
app = Flask(__name__)
CORS(app)
app.config["SECRET_KEY"] = secrets.token_hex(16)
DATABASE = 'database.db'
# Функції для роботи з базою даних
def get_db():
    db = getattr(g, '_database', None)
    if db is None:
        db = g._database = sqlite3.connect(DATABASE)
        db.row_factory = sqlite3.Row
    return db
@app.teardown_appcontext
def close_connection(exception):
    db = getattr(g, '_database', None)
    if db is not None:
        db.close()
def init_db():
    if not os.path.exists(DATABASE):
        with app.app_context():
            db = get_db()
            cursor = db.cursor()
            # Створення таблиць
            cursor.execute('''
            CREATE TABLE users (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                username TEXT UNIQUE NOT NULL,
                email TEXT UNIQUE NOT NULL,
                password TEXT NOT NULL,
                role TEXT NOT NULL,
                created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
            )
```

```
            ''')
            cursor.execute('''
            CREATE TABLE projects (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                title TEXT NOT NULL,
                description TEXT,
                file_path TEXT,
                user_id INTEGER NOT NULL,
                created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
                FOREIGN KEY (user_id) REFERENCES users (id)
            )
            ''')
            cursor.execute('''
            CREATE TABLE comments (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                content TEXT NOT NULL,
                user_id INTEGER NOT NULL,
                project_id INTEGER NOT NULL,
                created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
                FOREIGN KEY (user_id) REFERENCES users (id),
                FOREIGN KEY (project_id) REFERENCES projects (id)
            )
            ''')
            cursor.execute('''
            CREATE TABLE grades (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                value INTEGER NOT NULL,
                user_id INTEGER NOT NULL,
                project_id INTEGER NOT NULL,
                created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
                FOREIGN KEY (user_id) REFERENCES users (id),
                FOREIGN KEY (project_id) REFERENCES projects (id)
            )
            ''')
            cursor.execute('''
            CREATE TABLE messages (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                content TEXT NOT NULL,
                sender_id INTEGER NOT NULL,
                receiver_id INTEGER NOT NULL,
                created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
                FOREIGN KEY (sender_id) REFERENCES users (id),
                FOREIGN KEY (receiver_id) REFERENCES users (id)
            )
            ''')
            db.commit()
# Ініціалізація бази даних
init_db()
# Функції аутентифікації
def hash_password(password):
    return hashlib.sha256(password.encode()).hexdigest()
def authenticate(username, password):
    db = get_db()
    hashed_password = hash_password(password)
    user = db.execute(
        'SELECT * FROM users WHERE username = ? AND password = ?',
        (username, hashed_password)
    ).fetchone()
    return dict(user) if user else None
def require_auth(f):
    @wraps(f)  # Це збереже ім'я та метадані функції
    def decorated(*args, **kwargs):
        token = request.headers.get('Authorization')
        if not token:
            return jsonify({"error": "Необхідна авторизація"}), 401
```

```
        try:
            token = token.split()[1]  # Формат: "Bearer TOKEN"
            payload = jwt.decode(token, app.config['SECRET_KEY'], algorithms=['HS256'])
            user_id = payload['user_id']
            user_role = payload['role']
            db = get_db()
            user = db.execute('SELECT * FROM users WHERE id = ?', (user_id,)).fetchone()
            if not user:
                return jsonify({"error": "Користувача не знайдено"}), 404
            g.user = dict(user)
            g.user_role = user_role
        except jwt.ExpiredSignatureError:
            return jsonify({"error": "Термін дії токену закінчився"}), 401
        except (jwt.InvalidTokenError, Exception) as e:
            return jsonify({"error": "Недійсний токен"}), 401
        return f(*args, **kwargs)
    return decorated
# Маршрути аутентифікації
@app.route('/api/login', methods=['POST'])
def login():
    data = request.get_json()
    username = data.get('username')
    password = data.get('password')
    user = authenticate(username, password)
    if user:
        token = jwt.encode({
            'user_id': user['id'],
            'role': user['role'],
            'exp': datetime.datetime.utcnow() + datetime.timedelta(hours=24)
        }, app.config['SECRET_KEY'], algorithm='HS256')
        return jsonify({
            'token': token,
            'user_id': user['id'],
            'role': user['role'],
            'username': user['username']
        })
    return jsonify({"error": "Невірний логін або пароль"}), 401
@app.route('/api/register', methods=['POST'])
def register():
    data = request.get_json()
    username = data.get('username')
    email = data.get('email')
    password = data.get('password')
    role = data.get('role', 'student')  # За замовчуванням - студент
    if not all([username, email, password, role]):
        return jsonify({"error": "Всі поля обов'язкові"}), 400
    if role not in ['student', 'teacher', 'admin']:
        return jsonify({"error": "Невірна роль"}), 400
    db = get_db()
    # Перевірка на існуючого користувача
    existing_user = db.execute(
        'SELECT * FROM users WHERE username = ? OR email = ?',
        (username, email)
    ).fetchone()
    if existing_user:
        return jsonify({"error": "Користувач з таким ім'ям або email вже існує"}), 400
    hashed_password = hash_password(password)
    db.execute(
        'INSERT INTO users (username, email, password, role) VALUES (?, ?, ?, ?)',
        (username, email, hashed_password, role)
    )
    db.commit()
    return jsonify({"message": "Користувача успішно зареєстровано"}), 201
# Маршрути для роботи з проектами
@app.route('/api/projects', methods=['GET'])
```

```
def get_projects():
    db = get_db()
    projects = db.execute('''
        SELECT p.*, u.username as author,
        COUNT(DISTINCT c.id) as comments_count,
        AVG(g.value) as avg_grade
        FROM projects p
        JOIN users u ON p.user_id = u.id
        LEFT JOIN comments c ON p.id = c.project_id
        LEFT JOIN grades g ON p.id = g.project_id
        GROUP BY p.id
        ORDER BY p.created_at DESC
    ''').fetchall()
    return jsonify([dict(p) for p in projects])
@app.route('/api/projects/<int:project_id>', methods=['GET'])
def get_project(project_id):
    db = get_db()
    project = db.execute('''
        SELECT p.*, u.username as author
        FROM projects p
        JOIN users u ON p.user_id = u.id
        WHERE p.id = ?
    ''', (project_id,)).fetchone()
    if not project:
        return jsonify({"error": "Проект не знайдено"}), 404
    comments = db.execute('''
        SELECT c.*, u.username as author
        FROM comments c
        JOIN users u ON c.user_id = u.id
        WHERE c.project_id = ?
        ORDER BY c.created_at
    ''', (project_id,)).fetchall()
    grades = db.execute('''
        SELECT g.*, u.username as teacher
        FROM grades g
        JOIN users u ON g.user_id = u.id
        WHERE g.project_id = ? AND u.role = 'teacher'
    ''', (project_id,)).fetchall()
    result = dict(project)
    result['comments'] = [dict(c) for c in comments]
    result['grades'] = [dict(g) for g in grades]
    return jsonify(result)
@app.route('/api/projects', methods=['POST'])
@require_auth
def create_project():
    data = request.get_json()
    title = data.get('title')
    description = data.get('description', '')
    file_path = data.get('file_path', '')
    if not title:
        return jsonify({"error": "Назва проекту обов'язкова"}), 400
    db = get_db()
    db.execute(
        'INSERT INTO projects (title, description, file_path, user_id) VALUES (?, ?, ?, ?)',
        (title, description, file_path, g.user['id'])
    )
    db.commit()
    return jsonify({"message": "Проект успішно створено"}), 201
@app.route('/api/projects/<int:project_id>', methods=['PUT'])
@require_auth
def update_project(project_id):
    db = get_db()
    project = db.execute('SELECT * FROM projects WHERE id = ?', (project_id,)).fetchone()
    if not project:
        return jsonify({"error": "Проект не знайдено"}), 404
```

```
    # Перевірка прав доступу (власник або адмін)
    if project['user_id'] != g.user['id'] and g.user_role != 'admin':
        return jsonify({"error": "Немає дозволу на редагування проекту"}), 403
    data = request.get_json()
    title = data.get('title')
    description = data.get('description')
    file_path = data.get('file_path')
    if title:
        db.execute('UPDATE projects SET title = ? WHERE id = ?', (title, project_id))
    if description is not None:
        db.execute('UPDATE projects SET description = ? WHERE id = ?', (description,
project_id))
    if file_path:
        db.execute('UPDATE projects SET file_path = ? WHERE id = ?', (file_path, project_id))
    db.commit()
    return jsonify({"message": "Проект успішно оновлено"})
@app.route('/api/projects/<int:project_id>', methods=['DELETE'])
@require_auth
def delete_project(project_id):
    db = get_db()
    project = db.execute('SELECT * FROM projects WHERE id = ?', (project_id,)).fetchone()
    if not project:
        return jsonify({"error": "Проект не знайдено"}), 404
    # Перевірка прав доступу (власник або адмін)
    if project['user_id'] != g.user['id'] and g.user_role != 'admin':
        return jsonify({"error": "Немає дозволу на видалення проекту"}), 403
    # Видалення всіх пов'язаних записів
    db.execute('DELETE FROM comments WHERE project_id = ?', (project_id,))
    db.execute('DELETE FROM grades WHERE project_id = ?', (project_id,))
    db.execute('DELETE FROM projects WHERE id = ?', (project_id,))
    db.commit()
    return jsonify({"message": "Проект успішно видалено"})
# Маршрути для роботи з коментарями
@app.route('/api/projects/<int:project_id>/comments', methods=['POST'])
@require_auth
def add_comment(project_id):
    db = get_db()
    project = db.execute('SELECT * FROM projects WHERE id = ?', (project_id,)).fetchone()
    if not project:
        return jsonify({"error": "Проект не знайдено"}), 404
    data = request.get_json()
    content = data.get('content')
    if not content:
        return jsonify({"error": "Текст коментаря обов'язковий"}), 400
    db.execute(
        'INSERT INTO comments (content, user_id, project_id) VALUES (?, ?, ?)',
        (content, g.user['id'], project_id)
    )
    db.commit()
    return jsonify({"message": "Коментар успішно додано"}), 201
# Маршрути для оцінювання проектів (тільки для викладачів)
@app.route('/api/projects/<int:project_id>/grade', methods=['POST'])
@require_auth
def grade_project(project_id):
    if g.user_role not in ['teacher', 'admin']:
        return jsonify({"error": "Тільки викладачі можуть оцінювати проекти"}), 403
    db = get_db()
    project = db.execute('SELECT * FROM projects WHERE id = ?', (project_id,)).fetchone()
    if not project:
        return jsonify({"error": "Проект не знайдено"}), 404
    data = request.get_json()
    value = data.get('value')
    if not value or not isinstance(value, int) or value < 1 or value > 100:
        return jsonify({"error": "Оцінка повинна бути числом від 1 до 100"}), 400
    # Перевірка наявності оцінки від цього викладача
```

```
    existing_grade = db.execute(
        'SELECT * FROM grades WHERE user_id = ? AND project_id = ?',
        (g.user['id'], project_id)
    ).fetchone()
    if existing_grade:
        db.execute(
            'UPDATE grades SET value = ? WHERE user_id = ? AND project_id = ?',
            (value, g.user['id'], project_id)
        )
    else:
        db.execute(
            'INSERT INTO grades (value, user_id, project_id) VALUES (?, ?, ?)',
            (value, g.user['id'], project_id)
        )
    db.commit()
    return jsonify({"message": "Оцінка успішно збережена"})
# Маршрути для чату між користувачами
@app.route('/api/messages', methods=['GET'])
@require_auth
def get_messages():
    user_id = request.args.get('user_id')
    if not user_id:
        return jsonify({"error": "Необхідно вказати ID співрозмовника"}), 400
    db = get_db()
    messages = db.execute('''
        SELECT m.*, u.username as sender_name
        FROM messages m
        JOIN users u ON m.sender_id = u.id
        WHERE (m.sender_id = ? AND m.receiver_id = ?) OR (m.sender_id = ? AND m.receiver_id
= ?)
        ORDER BY m.created_at
    ''', (g.user['id'], user_id, user_id, g.user['id'])).fetchall()
    return jsonify([dict(m) for m in messages])
@app.route('/api/messages', methods=['POST'])
@require_auth
def send_message():
    data = request.get_json()
    receiver_id = data.get('receiver_id')
    content = data.get('content')
    if not all([receiver_id, content]):
        return jsonify({"error": "Усі поля обов'язкові"}), 400
    db = get_db()
    # Перевірка існування отримувача
    receiver = db.execute('SELECT * FROM users WHERE id = ?', (receiver_id,)).fetchone()
    if not receiver:
        return jsonify({"error": "Отримувача не знайдено"}), 404
    db.execute(
        'INSERT INTO messages (content, sender_id, receiver_id) VALUES (?, ?, ?)',
        (content, g.user['id'], receiver_id)
    )
    db.commit()
    return jsonify({"message": "Повідомлення успішно надіслано"}), 201
# Маршрути для адміністратора
@app.route('/api/users', methods=['GET'])
@require_auth
def get_users():
    if g.user_role != 'admin':
        return jsonify({"error": "Доступ заборонено"}), 403
    db = get_db()
    users = db.execute('SELECT id, username, email, role, created_at FROM users').fetchall()
    return jsonify([dict(u) for u in users])
@app.route('/api/chat/users', methods=['GET'])
@require_auth
def get_chat_users():
    db = get_db()
```

```
    users = db.execute('SELECT id, username, role FROM users').fetchall()
    return jsonify([dict(u) for u in users])
@app.route('/api/users/<int:user_id>', methods=['DELETE'])
@require_auth
def delete_user(user_id):
    if g.user_role != 'admin':
        return jsonify({"error": "Доступ заборонено"}), 403
    if user_id == g.user['id']:
        return jsonify({"error": "Неможливо видалити свій власний обліковий запис"}), 400
    db = get_db()
    user = db.execute('SELECT * FROM users WHERE id = ?', (user_id,)).fetchone()
    if not user:
        return jsonify({"error": "Користувача не знайдено"}), 404
    # Видалення всіх пов'язаних даних
    db.execute('DELETE FROM messages WHERE sender_id = ? OR receiver_id = ?', (user_id,
user_id))
    db.execute('DELETE FROM comments WHERE user_id = ?', (user_id,))
    db.execute('DELETE FROM grades WHERE user_id = ?', (user_id,))
    # Видалення проектів користувача
    projects = db.execute('SELECT id FROM projects WHERE user_id = ?', (user_id,)).fetchall()
    for project in projects:
        project_id = project['id']
        db.execute('DELETE FROM comments WHERE project_id = ?', (project_id,))
        db.execute('DELETE FROM grades WHERE project_id = ?', (project_id,))
    db.execute('DELETE FROM projects WHERE user_id = ?', (user_id,))
    db.execute('DELETE FROM users WHERE id = ?', (user_id,))
    db.commit()
    return jsonify({"message": "Користувача успішно видалено"})
# Маршрути для особистого кабінету
@app.route('/api/profile', methods=['GET'])
@require_auth
def get_profile():
    return jsonify(g.user)
@app.route('/api/profile', methods=['PUT'])
@require_auth
def update_profile():
    data = request.get_json()
    email = data.get('email')
    current_password = data.get('current_password')
    new_password = data.get('new_password')
    db = get_db()
    # Оновлення email
    if email and email != g.user['email']:
        # Перевірка на унікальність
        existing = db.execute('SELECT * FROM users WHERE email = ? AND id != ?',
                              (email, g.user['id'])).fetchone()
        if existing:
            return jsonify({"error": "Email вже використовується"}), 400
        db.execute('UPDATE users SET email = ? WHERE id = ?', (email, g.user['id']))
    # Зміна пароля
    if current_password and new_password:
        if hash_password(current_password) != g.user['password']:
            return jsonify({"error": "Неправильний поточний пароль"}), 400
        db.execute('UPDATE users SET password = ? WHERE id = ?',
                   (hash_password(new_password), g.user['id']))
    db.commit()
    return jsonify({"message": "Профіль успішно оновлено"})
if __name__ == '__main__':
    app.run(debug=True)
```